

# **ESTRUCTURA DE DATOS**



**PROGRAMA INGENIERIA DE SOFTWARE**  
**Modalidad de Educación a Distancia**  
**MODULO - ESTRUCTURA DE DATOS**



**ESTRUCTURA DE DATOS**  
**Programas de Educación a Distancia**  
**Universidad de Cartagena**  
**2017; [N°] Pág.; 21.5 X 27.9 cm**

© Bubok Publishing S.L., 2017  
1ª edición  
ISBN:  
Impreso en Colombia  
Impreso por...

**ESTRUCTURAS DE DATOS**

No está permitida la reproducción total o parcial de este libro o por cualquier medio o método de éste sin previa autorización de **la Universidad de Cartagena**, del Autor(a) y la **Empresa Editorial**. Ni su tratamiento informático, ni la transmisión de ninguna forma o por cualquier medio, ya sean electrónicos, mecánicos, por fotocopia, por registro u otros métodos, sin el permiso previo y por escrito de los titulares del Copyright.

DERECHOS RESERVADOS © 2017, respecto a la primera edición en español, por



Versión del Módulo: 1 - Página 2 de 76  
**Facultad de Ingeniería - Programa de Ingeniería del Software Mod. a Distancia**  
Avenida del Consulado #Calle 30 No. 48 – 152,  
Edificio Facultad de Ingeniería Tercer Piso  
Teléfono: (575) 6752040, (5) 6752024 ext. 208 Fax: 6752040 – Apartado Aéreo 1382  
[www.unicartagena.edu.co](http://www.unicartagena.edu.co) - Cartagena de Indias D. T. y C. - Colombia

**PROGRAMA INGENIERIA DE SOFTWARE**  
**Modalidad de Educación a Distancia**  
**MODULO - ESTRUCTURA DE DATOS**



**CONTENIDO**



	<b>Pág.</b>
1. Unidad 1: INTRODUCCIÓN A LAS ESTRUCTURAS DE DATOS.	6
1.1. Presentación de la unidad.	6
1.2. Mapa Conceptual.	9
1.3. Evaluación de Pre saberes.	10
1.4. Lección 1: ESTRUCTURA DE DATOS	10
1.4.1. Introducción a la Estructura de Datos	10
1.4.2. ¿Qué es una estructura de datos?	11
1.4.3. Tipos de Datos – Clasificación	12
1.5. Lección 2: ARRAY (ARREGLOS)	18
1.5.1. Objetivos	18
1.5.2. Introducción	18
1.5.3. ¿Que son los Arreglos?	19
1.5.4. ¿Cómo se Clasifican los Arreglos?	21
1.6. Lección 3. ¿Que son los Arreglos Unidimensionales (Vectores)?	21
□ Declaración de Arreglos	22
□ Construcción de Arreglos	25
□ Inicializar Arreglos	27
1.6.1. Tamaño de los Arreglos, atributo length	27
1.6.2. Copiar y Comparar Arreglos en Java?	31
1.6.3. La clase arrays del api de java. Equals, copyof, fill.	33



**PROGRAMA INGENIERIA DE SOFTWARE**  
**Modalidad de Educación a Distancia**  
**MODULO - ESTRUCTURA DE DATOS**



1.6.4. Rellenar un array con un valor u objeto. Método fill de la clase arrays	39
1.6.5. Recorrer un Array Unidimensional	41
1.6.6. Recorrer un Array en java con for-each. Bucle for para colecciones	44
1.6.7. ¿Por qué se clasifica un arreglo?	46
1.6.8. Utilización de dispositivos de almacenamiento externo.	48
1.6.8.1. Archivos de texto.	48
1.6.8.2. Principales consideraciones semánticas.	51
1.6.8.3. Sintaxis de las principales operaciones relacionadas con los Archivos de texto.	52
1.6.8.4. Archivos binarios.	53
1.6.9. ¿Por qué se ordena un arreglo?	58
1.6.9.1. Ordenamiento por el Método de Burbuja	58
1.7. Lección 4. ¿Que son los Arreglos Bidimensionales (Matrices)?	66
REFERENCIAS BIBLIOGRÁFICAS	75





## **UNIDAD I.**

### **Introducción a las Estructuras de Datos.**

---

#### **1. Unidad 1: INTRODUCCIÓN A LAS ESTRUCTURAS DE DATOS.**

##### **1.1. Presentación de la unidad.**

#### **INTRODUCCIÓN A LAS ESTRUCTURAS DE DATOS**

En esta unidad, explicaremos la interfaz y vectores de realización básica, estrategia dinámica gestión del espacio, tecnología única, la búsqueda binaria, ordenamiento de burbuja, ordenamiento por mezcla y así sucesivamente. El contenido se aplica a menudo en la práctica, sino también que deben dominar los conceptos básicos.

#### **SABER CONCEPTUAL : (Teorías, Definiciones)**

- 1.1. Introducción a la Estructura de Datos
- 1.2. Estructuras de Datos Estáticas.
  - 1.2.1. Arreglos Unidimensionales
  - 1.2.2. Arreglos Bidimensionales
  - 1.2.3. Ejercicios Resueltos
  - 1.2.4. Aplicaciones
- 1.3. Métodos de Ordenación y Búsquedas

**PROGRAMA INGENIERIA DE SOFTWARE**  
**Modalidad de Educación a Distancia**  
**MODULO - ESTRUCTURA DE DATOS**



- 1.3.1. Ordenación Interna y Búsquedas internas
- 1.3.2. Ejercicios Resueltos
- 1.3.3. Aplicaciones

**SABER HACER (Aplicaciones)**

- ☐ Diferenciar las estructuras de datos más utilizadas.
- ☐ Entender los conceptos de apuntadores para crear estructuras de datos.
- ☐ Comprender la Asignación de Memoria usando apuntadores.
- ☐ Realizar Operaciones (Buscar, adicionar o eliminar, pegar, romper listas, ordenar, copiar nodos) de una Lista.
- ☐ Implementar estructuras de datos en lenguajes como Java.
- ☐ Desarrollar aplicaciones usando algunas de las Estructuras definidas.

**SABER SER (Valores)**

Solidaridad

Amor por el saber

Actitud crítica.

Cultura por el trabajo individual y en equipo.

Creatividad e imaginación.

Adaptación al cambio de trabajo en otro idioma.

Cumplimiento de las consultas y ejercicio.

Capacidad de razonamiento e interpretación

Responsabilidad y Autonomía

Constancia para llegar al resultado.



**PROGRAMA INGENIERIA DE SOFTWARE**  
**Modalidad de Educación a Distancia**  
**MODULO - ESTRUCTURA DE DATOS**



EVALUACION DEL APRENDIZAJE
<b>EVIDENCIA DE CONOCIMIENTO (Evidencia oral o escrita)</b>
<ul style="list-style-type: none"><li>• Trabajos en grupo y Exposición</li><li>• Laboratorios</li><li>• Examen escrito</li></ul>
<b>EVIDENCIA DE DESEMPEÑO (Evidencia del hacer)</b>
Resolución de Ejercicios Protocolos
<b>EVIDENCIA DE PRODUCTO (Evidencia Tangible)</b>
Trabajo de Investigación  Proyecto de clase

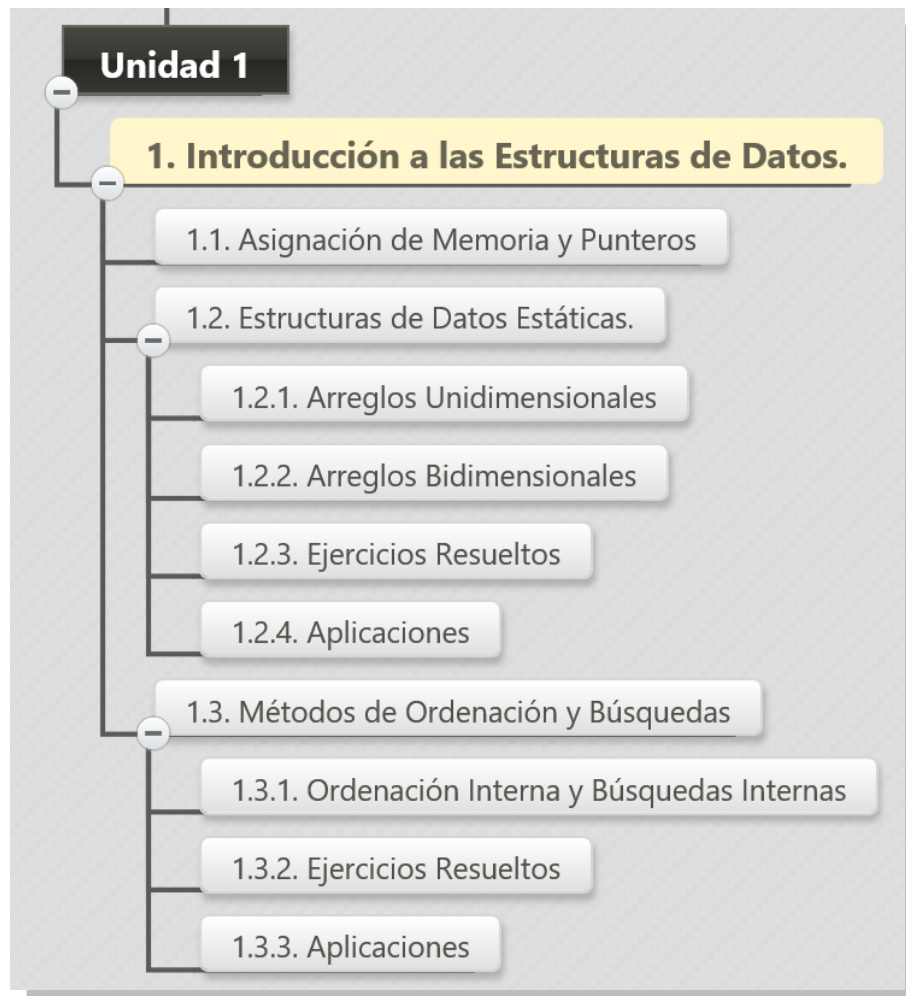




**PROGRAMA INGENIERIA DE SOFTWARE**  
**Modalidad de Educación a Distancia**  
**MODULO - ESTRUCTURA DE DATOS**



**1.2. Mapa Conceptual.**



**Ilustración 1. Mapa Conceptual Unidad 1<sup>1</sup>**

<sup>1</sup> Fuente: Autoría Propia. Ref. Web: <https://www.mindmeister.com/es/580337973/m-dulo-estructura-de-datos>



**PROGRAMA INGENIERIA DE SOFTWARE**  
**Modalidad de Educación a Distancia**  
**MODULO - ESTRUCTURA DE DATOS**



**1.3. Evaluación de Pre saberes.**

**AUTOEVALUACIÓN**

- ¿Qué entiende usted por estructura de datos?
- ¿Explique con sus palabras que son estructuras datos estáticas y dinámicas?
- ¿Dé ejemplos de estructuras de datos dinámicas y estáticas?
- ¿Pensando en optimizar al máximo los recursos del sistema, que estructura de datos entre estáticas y dinámicas utilizaría usted?. Explique la respuesta.



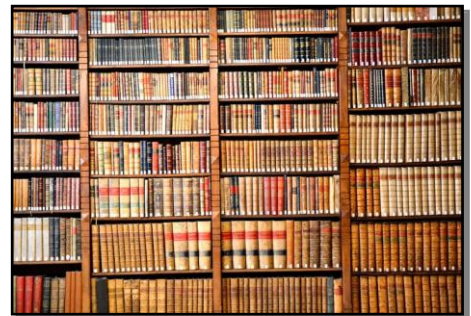
**1.4. Lección 1: ESTRUCTURA DE DATOS**

**1.4.1. Introducción a la Estructura de Datos**



Podrías imaginar a la hora de buscar un libro en una biblioteca para leerlo, nos encontremos con que ésta no se encuentra ordenada, esto sería un gran caos y nos demoraría horas para encontrar el libro, pues bien, esta pequeña introducción nos ayudara a comprender el concepto de estructura de datos.

Las estructuras de datos nos permiten almacenar, manipular y ordenar los datos, los cuales son materia prima en cualquier sistema de información. En el punto anterior, podemos verla como el método que nos ayudará a organizar los libros de forma adecuada ahorrándonos tiempo en la búsqueda.



**PROGRAMA INGENIERIA DE SOFTWARE**  
**Modalidad de Educación a Distancia**  
**MODULO - ESTRUCTURA DE DATOS**



**1.4.2. ¿Qué es una estructura de datos?**

- A. **Primera Definición:** Una estructura de datos es un modelo matemático o lógico de una organización particular de datos.
- B. **Segunda Definición:** Una estructura de datos es una colección de datos que se caracterizan por la FORMA en que estos se ORGANIZAN, y por las OPERACIONES que se pueden definir sobre dichos datos.
- C. **Tercera Definición:** En programación, una estructura de datos es una forma de organizar un conjunto de datos elementales (Un dato elemental es la mínima información que se tiene en un sistema) con el objetivo de facilitar su manipulación.

Una Estructura de Datos es simplemente “Una Organización intencional de datos”

Las palabras:

- **Organización:** significa colección de datos. Ejemplo: tomemos los datos de un mismo tipo (libros) y lo coloquemos en un mismo lugar.
- **Intencional de datos** quiere decir que vamos a tener una intensión con los datos, es decir, no vamos a mezclar datos de diferentes tipos, sino, mantenerlos organizados para almacenarlos y hacer una excelente búsqueda luego de esta información.

Su propósito es permitir un eficiente almacenamiento y recuperación de los datos.

En general, la estructuración de los datos es una parte inherente a la solución del problema.

Una estructura de datos es una agrupación de éstos que se trata como una unidad en su conjunto.

Las estructuras de datos pueden ser homogéneas (todos los datos son del mismo tipo) o heterogéneas (constituidas por datos de tipos diferentes).

Las estructuras de datos homogéneas más representativas son los vectores, las tablas y, en general, las matrices n-dimensionales. El ejemplo más representativo de estructuras de datos heterogéneas son los registros



**PROGRAMA INGENIERIA DE SOFTWARE**  
**Modalidad de Educación a Distancia**  
**MODULO - ESTRUCTURA DE DATOS**



Desde otro punto de vista puede hablarse de estructuras de datos estáticas y dinámicas.

Una estructura estática se caracteriza porque su tamaño es conocido a priori (antes de la ejecución del programa). En consecuencia, en el código del programa se declaran variables de tipo estático y en la compilación se reserva en memoria el espacio necesario para ellas.

Por el contrario, las estructuras de datos dinámicas no tienen un tamaño predefinido y la memoria utilizada para almacenarlas se reserva o libera, en tiempo de ejecución, según se requiera.

### **1.4.3. Tipos de Datos – Clasificación**

Los tipos de datos que utilizaremos en este módulo y usando como lenguaje de programación, java son:



**PROGRAMA INGENIERIA DE SOFTWARE**  
**Modalidad de Educación a Distancia**  
**MODULO - ESTRUCTURA DE DATOS**



Universidad  
de Cartagena  
Fundada en 1827



Accreditación Institucional de Alta Calidad  
Resolución 2583 del 26 de febrero de 2014. Ministerio de Educación Nacional



Ilustración 2. Tipos de Datos – Clasificación<sup>2</sup>

<sup>2</sup> Fuente: Autoría Propia. Ref. Web: <https://www.mindmeister.com/629689949/estructura-de-datos-tipos-de-datos-clasificaci-n>



**PROGRAMA INGENIERIA DE SOFTWARE**  
**Modalidad de Educación a Distancia**  
**MODULO - ESTRUCTURA DE DATOS**



#### 1.4.1. Estructuras de datos dinámicas

Se dice que una estructura de datos es dinámica cuando inicialmente (en el momento de la compilación) no tiene espacio asignado para almacenar información. **Durante la ejecución** del programa el sistema (en tiempo de ejecución, *run time*) asigna y libera espacio en memoria, en función de las necesidades.

Los datos se almacenan en estructuras de datos independientes en alguna dirección de memoria. Cada estructura está conectada a una o más estructuras del mismo tipo mediante un puntero (nexo hacia alguna dirección de memoria). El tamaño de las estructuras dinámicas es variable, depende de la cantidad de datos almacenados. La Ilustración 3 muestra un ejemplo.

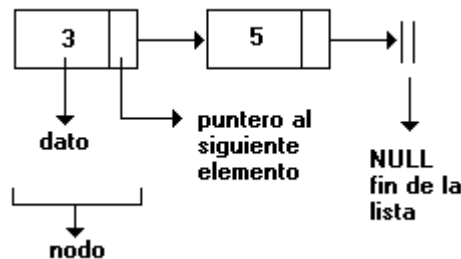


Ilustración 3. Estructura de datos dinámica.<sup>3</sup>

Los temas (Listas), (Árboles) y (Grafos) describen distintos tipos de Estructuras de Datos Dinámicas.

En algunos lenguajes de programación, para permitir la implementación de estructuras de datos dinámicas es necesario un tipo de datos especial, denominado **puntero** (*pointer*).

El concepto de **puntero** (*pointer*) hace referencia a una variable cuyo contenido es la dirección de otra variable (**nodo**) que realmente contiene el propio dato que se emplea en el programa<sup>4</sup>.

<sup>3</sup> Fuente: Ref. Web: <http://www.angelfire.com/my/jimena/estructuras/materia1.html>

<sup>4</sup> Este concepto toma su soporte físico en el mecanismo de direccionamiento indirecto tal como se maneja en los lenguajes de bajo nivel.





**PROGRAMA INGENIERIA DE SOFTWARE**  
**Modalidad de Educación a Distancia**  
**MODULO - ESTRUCTURA DE DATOS**



Ilustración 4. muestra un modelo de funcionamiento.

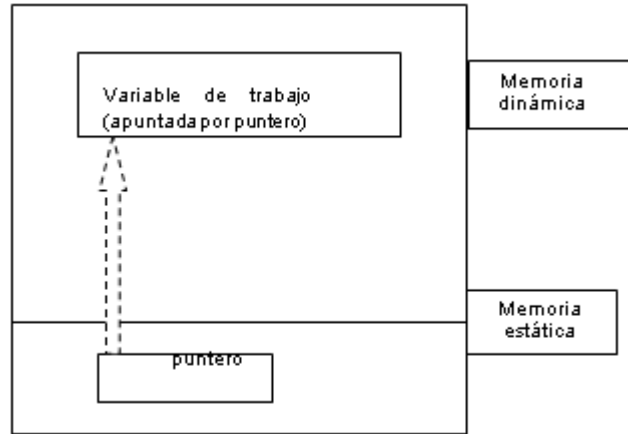


Ilustración 4. *Punteros y variables de trabajo.*

El interés del uso de punteros en lenguajes de programación de alto nivel reside en que constituyen la base para la generación de estructuras de datos dinámicas. Esto es, que a lo largo de la ejecución de un programa, y como consecuencia de las sucesivas operaciones de inserción y eliminación, el sistema en tiempo de ejecución (*run time*) reserva y libera respectivamente unidades (nodos) de una región de memoria destinada a uso dinámico (a diferencia de la que se reserva en el momento de la compilación que contiene el código máquina y el espacio requerido por las variables estáticas). Este espacio (estático) se mantiene inalterado durante toda la ejecución del programa.

La utilización de esta posibilidad requiere que el lenguaje de programación disponga de operaciones que soliciten (de forma transparente al programador) espacio en la memoria dinámica para crear nuevos nodos, así como la contraria, para liberar espacio correspondiente a nodos que ya no se necesitan y poder disponer, en consecuencia, de dicho espacio para uso posterior. En Java se utiliza la sintaxis:

**<tipo> <variable> = new <tipo>;** para crear un nodo

La variable puntero (en Java se denominan referencias), se almacena en la memoria estática y su tamaño es fijo. No así los nodos que pueden ser de diferente naturaleza. Es decir que en la declaración de un nodo debe hacerse referencia al tipo de dato al que apunta para que cuando se cree un nodo se reserve con el tamaño necesario. Por ejemplo, al ejecutar el código en Java:



**PROGRAMA INGENIERIA DE SOFTWARE**  
**Modalidad de Educación a Distancia**  
**MODULO - ESTRUCTURA DE DATOS**



```
char [ ] puntVector = new char [100];
```

se declara una variable (*puntVector*) de tal forma que, al realizar la operación *new* se reserve el espacio necesario para almacenar un vector de 100 caracteres.

Además de la operación ya indicada de reserva (*new*), las referencias admiten las siguientes operaciones:

- Asignación:  $\langle puntero1 \rangle = \langle puntero2 \rangle$ . La referencia  $\langle puntero2 \rangle$  se copia en  $\langle puntero1 \rangle$ , o dicho en otros términos:  $\langle puntero1 \rangle$  deja de apuntar al nodo a que apuntaba previamente para pasar a apuntar al nodo apuntado por  $\langle puntero2 \rangle$ . Una vez ejecutada la operación ambos punteros apuntan al mismo nodo<sup>41</sup>.
- Comparación:  $\langle puntero1 \rangle == \langle puntero2 \rangle$ . Devuelve un valor booleano en función de que ambas referencias sean iguales o no, o dicho en otros términos: que apunten o no al mismo nodo.

Aunque, como se ha indicado, una variable de tipo puntero apunta a una zona de memoria (nodo) existe una situación excepcional consistente en no apuntar a ninguno. En Java se utiliza para esto la constante *null*.

Sobre las variables referencia, se realizan las operaciones propias del tipo de datos a que pertenezcan.

Las estructuras de datos dinámicas también se pueden clasificar de la siguiente forma:

● **Lineales**. Cada estructura se relaciona únicamente con una sola estructura. Por ejemplo, la lista enlazada mostrada en la Ilustración 5.

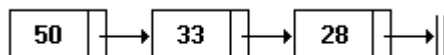


Ilustración 5. Estructura de datos lineal.<sup>5</sup>

● **No lineales**. Se relacionan con más de una estructura. Por ejemplo, el árbol binario mostrado en la Ilustración 6.

<sup>5</sup> Fuente: Ref. Web: <http://www.angelfire.com/my/jimena/estructuras/materia1.html>



**PROGRAMA INGENIERIA DE SOFTWARE**  
**Modalidad de Educación a Distancia**  
**MODULO - ESTRUCTURA DE DATOS**

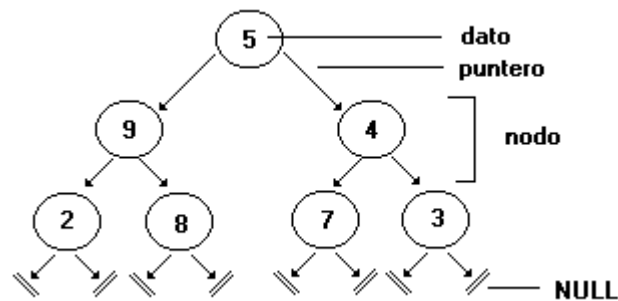


Ilustración 6. Estructura de datos no lineal<sup>6</sup>

#### 1.4.2. Estructuras de datos estáticas.

Una estructura **estática** se caracteriza porque su tamaño es conocido a priori (antes de la ejecución del programa). En consecuencia, en el código del programa se declaran variables de tipo estático y en la compilación se reserva en memoria el espacio necesario para ellas.

Los datos se almacenan en la memoria en forma secuencial (uno a continuación del otro, como lo muestra la Ilustración 7), dentro de una misma estructura de datos. El tamaño de las estructuras estáticas es fijo, se señala al declararla en el programa, por lo cual la cantidad de datos a ingresar es limitada.

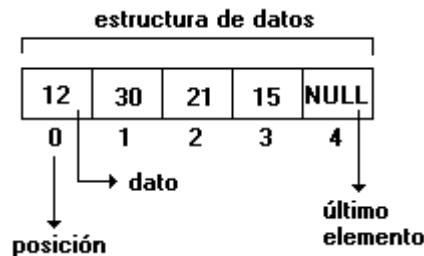


Ilustración 7. Estructura de datos estática.<sup>7</sup>

<sup>6</sup> Fuente: Ref. Web: <http://www.angelfire.com/my/jimena/estructuras/materia1.html>

<sup>7</sup> Fuente: Ref. Web: <http://www.angelfire.com/my/jimena/estructuras/materia1.html>

<sup>41</sup> Este tipo de operaciones deberá hacerse con especial cuidado pues, si no se han tomado previamente las precauciones oportunas, podría perderse la información almacenada en el nodo inicialmente apuntado por <puntero1> (se perderá si no tenemos otra referencia a esa información)

**PROGRAMA INGENIERIA DE SOFTWARE**  
**Modalidad de Educación a Distancia**  
**MODULO - ESTRUCTURA DE DATOS**



## **1.5. Lección 2: ARRAY (ARREGLOS)**

### **1.5.1. Objetivos**

En esta lección aprenderá a:

- Diferenciar entre un tipo simple y un tipo estructurado.
- Aprenderá a diferenciar entre un dato primitivo y dato de tipo estructurado.
- Acceder a los elementos de un vector y de una matriz en java.
- Inicializar elementos de un arrays
- Declarar una variable de tipo arreglo (array)
- Definir un arreglo con el operador new
- Acceder a los elementos de un arreglo.
- Pasar arrays a un método y distinguir paso por valor y paso por referencia.
- Conocer algoritmos sencillos de ordenación de arrays.
- Conocer la clase predefinida ArrayList

### **1.5.2. Introducción**

Sabemos que los datos de un programa se almacenan en las variables y por lo general toma los espacios de memoria al azar pero en el caso en que necesitemos los datos, estos deben ser del mismo tipo llamados elementos, y pueden ser datos simples de java, o de una clase previamente declarada como tal.

Un Arreglo tiene la misma definición de un Arreglo de datos Primitivos, estos difieren en la instancia que poseen, más claramente, estos se crean en base a una clase ya existente y definida con sus atributos y métodos correspondientes, recordando: Una clase es la definición de un objeto, ya que esta posee los atributos (características) y funciones (métodos) que describen un objeto. La definición de un Arreglo de Objetos es la misma que un Arreglo de datos Primitivos, su sintaxis es de la siguiente forma:

Clase nombreArreglo[] = new clase [numeroElementos];

Los arreglos se crean con el operador new seguido del tipo y número de elementos y se puede acceder al número de elementos de un arreglo con la variable miembro implícita length (por ejemplo, v.length).

Se accede a los elementos de un arreglo con los corchetes [] y un índice que varía de = a Length-1.

Se pueden crear arreglos de objetos de cualquier tipo. En principio un arreglo de objetos es un arreglo de referencias que hay que completar llamando al operador new.



**PROGRAMA INGENIERIA DE SOFTWARE**  
**Modalidad de Educación a Distancia**  
**MODULO - ESTRUCTURA DE DATOS**



Los elementos de un arreglo se inicializan al valor por defecto del tipo correspondiente (cero para valores numéricos, la cadena vacía para Strings, false para boolean, null para referencia).

Los arreglos se pueden inicializar con valores entre llaves {...}.

Si se igualan dos referencias a un arreglo y no se copia el arreglo, sino que se tiene un arreglo con dos nombres, apuntando al mismo y único objeto.

Un Arreglo por lo general, forma parte de los valores que hacen referencia implícitamente con la ayuda de los valores de índice. Es por ello que con el fin de acceder a los valores almacenados en un Array; se están tomando la ayuda de índices. Supongamos que tenemos un Array que contiene números enteros "n", entonces vamos a tener su primer elemento está indexado con el valor "0" y el último número entero será referenciado por "n-1" valor indexado.

Ahora supongamos que un Array que se compone de 12 elementos con cada elemento es la celebración de un valor distinto. Tendremos el primer elemento que hace referencia por un [0], es decir, el primer valor del índice. Hemos llenado los 12 valores distintos en el Array de cada referencia como:

```
a[0]=1
a[1]=2
...
a[n-1]=n
...
a[11]=12
```

### 1.5.3. ¿Que son los Arreglos?

Es un conjunto **ordenado** y **finito** de elementos **homogéneos**. Es decir, sus características básicas son:

- "**Finito**", porque se requiere definir el tamaño del array (definir el tamaño antes de ser utilizado).  
Ej. : El array Notas que almacena las notas de los 25 alumnos de una clase es de tamaño 25.
- "**Homogéneos**", porque todos los elementos del array son del **mismo tipo**.



PROGRAMA INGENIERIA DE SOFTWARE  
Modalidad de Educación a Distancia  
MODULO - ESTRUCTURA DE DATOS

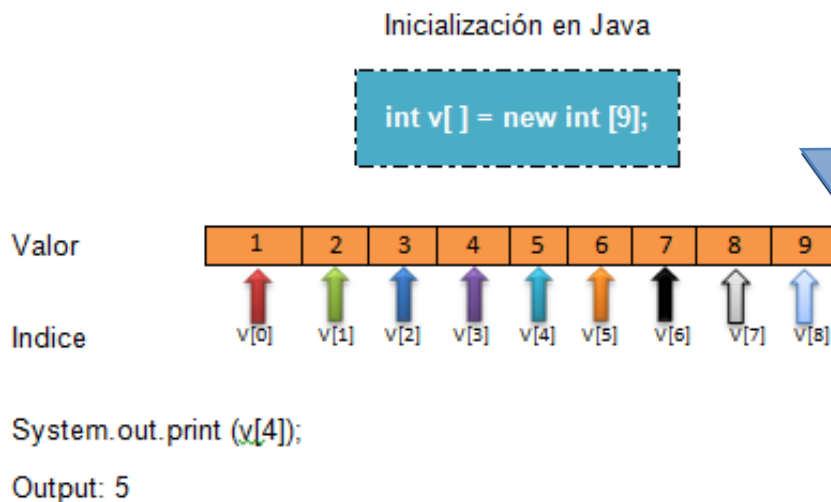


Ej.: en el array Notas, todas las notas almacenadas son de tipo entero.

- "**Ordenado**", porque se pueden identificar a cada elemento del array por la posición que ocupan: el primero, el segundo, el tercero,..., el n-esimo, etc.  
Ej.: en el array Notas, la nota del tercer alumno de la clase (puede ser en orden alfabético), ocupa la posición 3.

Ejemplo:

- Para declarar un arreglo tiene que indicar su tipo, un nombre único y la cantidad de elementos que va a contener.
- Para crear un arreglo en Java se debe utilizar el operador new.



Un array (arreglo) en Java es una estructura de datos que nos permite almacenar un conjunto de datos de un mismo tipo. El tamaño de los arrays se declara en un primer momento y no puede cambiar luego durante la ejecución del programa, como sí puede hacerse en otros lenguajes. Veremos ahora cómo declarar arrays estáticos de una dimensión.



Ilustración 1.Representación de la Estructura de un Arreglo<sup>8</sup>

<sup>8</sup> Fuente: Autoría Propia.



PROGRAMA INGENIERIA DE SOFTWARE  
Modalidad de Educación a Distancia  
MODULO - ESTRUCTURA DE DATOS



1.5.4. ¿Cómo se Clasifican los Arreglos?

Arreglo  
Unidimensional  
(Array Lineal  
o **Vector** )



En matemática es conocido como Vector. *Ejem: Cantidad de Goles anotados por el equipo colombiano en cada uno de los 5 partidos del sudamericano 2015.*

 Tamaño  
5

Arreglo  
**Bidimensional**  
(Array  
Bidimensional  
o **Matriz** )



En matemática es conocido como Matriz, o en base de datos como tabla. *Ejem: Los sueldos de 10 empleados en cada uno de los meses de Enero a Junio 2016.*

 Tamaño  
10x6

Arreglo  
**Multidimensional**  
(n - dimensional)



*Ejem: Estados (libre u ocupado) de las 10 aulas en cada uno de los 4 pisos de los 5 pabellones.*


 Tamaño  
10x4x5

Ilustración 9. Clasificación de Arreglo

1.4.2.1. Estructuras de datos homogéneas.

En las estructuras de datos homogéneas todos los datos son del mismo tipo. Como ejemplo veremos los **vectores**, las **tablas** y, en general, las **matrices**  $n$ -dimensionales.

1.6. Lección 3. ¿Que son los Arreglos Unidimensionales (Vectores)?

Un array (arreglo) es una estructura de datos que contiene una colección de datos del mismo tipo, estas son usadas como contenedores que almacenan uno o más datos relacionados, en lugar de declarar cada dato de manera independiente.

Por medio de los arreglos veremos cómo crear un elemento que nos permite definir una “variable” que contenga diferentes datos del mismo tipo asociados al mismo identificador.



**PROGRAMA INGENIERIA DE SOFTWARE**  
**Modalidad de Educación a Distancia**  
**MODULO - ESTRUCTURA DE DATOS**



Aunque ya se mencionó que los arreglos unidimensionales también son conocidos como vectores, es muy común que estos sean conocidos solo como arreglos simplemente refiriéndose a ellos como los de una sola dimensión, se componen de una fila y una o más columnas, las cuales representan espacios de memoria donde se pueden almacenar datos.

Para poder trabajar con arreglos, se debe crear el arreglo definiendo cual es el tipo de datos que va a contener y cuál es el tamaño del mismo (cantidad de datos que puede almacenar), de ese modo si definimos que el arreglo va a ser de tipo byte, solo podrá almacenar datos de tipo byte, no se puede mezclar en dicho arreglo datos int, short o long por ejemplo.

Un arreglo es una secuencia de datos del mismo tipo:

- Los datos se llaman elementos del arreglo y se numeran 0, 1, 2, ...
- Estos números localizan al elemento dentro del arreglo y se denominan índices.
- En Java, los índices del arreglo empiezan con 0 y terminan con el tamaño del arreglo -1.
- Si el arreglo tiene n elementos, se denotan como a[0], a[1], ... a[n-1]

La creación de un arreglo se realiza en 3 pasos básicos: Declaración, construcción e inicialización

### **Declaración de Arreglos**

Se declara de modo similar a otros tipos de datos, excepto que se debe indicar al compilador que es un arreglo y esto se hace con corchetes.

```
tipo_dato nombre_array[ ]; // indica que todos los identificadores  
son arreglos de tipo
```

```
tipo_dato [ ] nombre_array; // sólo es arreglo el identificador al que  
le siguen los [ ].
```

Donde **tipo\_dato** define el tipo de dato de cada uno de los valores que puede contener el arreglo.



**PROGRAMA INGENIERIA DE SOFTWARE**  
**Modalidad de Educación a Distancia**  
**MODULO - ESTRUCTURA DE DATOS**



**Ejemplo:**

Formas de declarar arreglos:

- 1) `char c [ ], j;` // `c` es un arreglo de tipo `char`; `j` es una variable del mismo tipo.
- 2) `int [ ] n, m;` // tanto `n` como `m` son arreglos unidimensionales de tipo `int`.
- 3) `Double [ ] x, h [ ], v;` // `x` y `v` son arreglos de tipo `double`; `h` es un arreglo con elementos de tipo `double`.



Java no permite indicar el número de elementos en la declaración de un arreglo, `int n[10];` // el compilador producirá un error.

El tipo de variable puede ser cualquiera de los admitidos por Java y que mostraremos a continuación. Ejemplos de otras formas de declaración e inicialización con valores por defecto de arrays usando todos los tipos de variables Java, serían:

- La sintaxis de declaración de arreglos en Java es:
  - o `tipo [ ] identificador`
  - o `tipo identificador [ ]`
- Ejemplos para declaración de un arreglo:
  - o `char cad[ ], p;`
  - o `double [ ] m, t[ ], x;`
  - o `int [ ] v, w;`
- Sintaxis para definir arreglo de un número determinado de elementos:
  - o `tipo nombreArreglo[ ] = new tipo [numeroDeElementos]`
  - o `tipo nombreArreglo[ ];`
  - o `nombreArreglo = new tipo[numeroDeElementos];`
- Ejemplos definir arreglo de número de elementos:
  - o `float notas = new float [26];`
  - o `int [ ] a;`
  - o `a = new int[10];`
  - o `byte[ ] edad = new byte[4];`
  - o `short[ ] edad = new short[4];`
  - o `int[ ] edad = new int[4];`
  - o `long[ ] edad = new long[4];`





**PROGRAMA INGENIERIA DE SOFTWARE**  
**Modalidad de Educación a Distancia**  
**MODULO - ESTRUCTURA DE DATOS**



- float[ ] estatura = new float[3];
- double[ ] estatura = new double[3];
- boolean[ ] estado = new boolean[5];
- char[ ] sexo = new char[2];
- String[ ] nombre = new String[2];
- En la declaración del arreglo no se permite indicar el número de elementos, así, int números [12] es erróneo

En caso de que queramos inicializarlos con valores propios, haremos esto:

- Para números enteros
  - int[ ] edad = {15, 32, 9, 11}; //Array de 4 elementos
  - De la misma forma procederíamos para los otros tipos de enteros: byte, short, long.
- Para números reales
  - double[ ] estatura = {1.63, 1.77, 1.80}; //Array de 3 elementos
  - De la misma forma procederíamos para el tipo float, pero teniendo en cuenta que los números deberán llevar al final la letra "f" o "F". Por ejemplo 1.63f o 1.63F.
- Para cadenas
  - String[ ] nombre = {"Mónica", "Carlos"}; //Array de 2 elementos
- Para caracteres
  - char[ ] sexo = {'m', 'f'}; //Array de 2 elementos
- Para booleanos
  - boolean[ ] = {true,false}; //Array de 2 elementos

Ejemplo:

Cuando creamos un array de nombre "a" y de dimensión "n" (int[ ] a = new int[n]) estamos creando n variables que son a[0], a[1], a[2], ..., a[n-1].

Los arrays se numeran desde el elemento cero, que sería el primer elemento, hasta el n-1 que sería el último elemento. Es decir, si tenemos un array de 5 elementos, el primer elemento sería el cero y el último elemento sería el 4. Esto conviene tenerlo en cuenta porque puede dar lugar a alguna confusión. Disponer de un valor con índice cero puede ser de utilidad en situaciones como considerar cada variable asociada a una hora del día, empezando a contar desde la hora cero hasta la 23 (total de 24 horas), cosa que es habitual en algunos países. En lugar de 1, 2, 3,..., 24 estaríamos usando 0, 1, 2, ..., 23.





**PROGRAMA INGENIERIA DE SOFTWARE**  
**Modalidad de Educación a Distancia**  
**MODULO - ESTRUCTURA DE DATOS**



Para acceder a un elemento específico utilizaremos los corchetes de la siguiente forma. Entendemos por acceso, tanto el intentar leer el elemento, como asignarle un valor.

```
arrayCaracteres[numero_elemento];
```

Por ejemplo, para acceder al tercer elemento lo haríamos de la siguiente forma:

```
// Lectura de su valor.  
char x = arrayCaracteres[2];  
  
// Asignación de un valor. Como se puede comprobar se pone el  
// número dos, que coincide con el tercer elemento. Ya que como  
// dijimos anteriormente el primer elemento es el cero.  
arrayCaracteres[2] = 'b';
```

El objeto array, aunque podríamos decir que no existe como tal, posee una variable, la cual podremos utilizar para facilitar su manejo.

### **Construcción de Arreglos**

Java considera que un arreglo es una referencia a un objeto; en consecuencia, para que realmente cree o instancie el arreglo, usa el operador *new* junto al tipo de los elementos del arreglo y su número.

Después de haber declarado el array se puede construir e inicializar de 2 maneras.

- **Forma 1:** la primera se usa cuando inicialmente no sabemos cuáles son los valores que va a contener el arreglo, ya que luego serán ingresados, se crea con la siguiente estructura:

```
tipo_dato nombre_array [ ];  
nombre_array = new tipo_dato[tamaño];
```

Ej. arregloDeEnteros = new int[5];



**PROGRAMA INGENIERIA DE SOFTWARE**  
**Modalidad de Educación a Distancia**  
**MODULO - ESTRUCTURA DE DATOS**



**Ejemplo Forma 1:**

```
String arregloA[]; //Declaración del arreglo  
arregloA=new String[4]; //Creación o construcción del arreglo
```

```
//Llenado del arreglo
```

```
arregloA[0]="hola"; //inicialización arreglo en la posición 0  
arregloA[1]="casa"; //inicialización arreglo en la posición 1  
arregloA[2]="perro"; //inicialización arreglo en la posición 2  
arregloA[3]="oso"; //inicialización arreglo en la posición 3
```

```
//Obteniendo información del arreglo
```

```
System.out.println("Valor arreglo en la posición 0: "+arregloA[0]);  
System.out.println("Valor arreglo en la posición 1: "+arregloA[1]);  
System.out.println("Valor arreglo en la posición 2: "+arregloA[2]);  
System.out.println("Valor arreglo en la posición 3: "+arregloA[3]);
```

- **Forma 2:** Esta forma se usa cuando sabemos con exactitud cuáles son los valores que va a contener el arreglo, aquí el proceso de construcción e inicialización se hace directo y se realiza de la siguiente manera:

`tipo_dato [ ] nombre_array = {valor1, valor2, valor3, valor4}`

Ej: `int[ ] arregloDeEnteros= {2, 3, 6, 8,3};`



**Ejemplo Forma 2:**

```
String arregloA[]; //Declaración del arreglo  
//Declaración, Inicialización y Creación del arreglo  
String nombres[]={ "Carlos", "Julian", "Cristian", "Miguel"};
```

```
//Obteniendo información del arreglo
```

```
System.out.println("Valor arreglo en la posición 0: "+nombres[0]);  
System.out.println("Valor arreglo en la posición 1: "+nombres[1]);  
System.out.println("Valor arreglo en la posición 2: "+nombres[2]);  
System.out.println("Valor arreglo en la posición 3: "+nombres[3]);
```



**PROGRAMA INGENIERIA DE SOFTWARE**  
**Modalidad de Educación a Distancia**  
**MODULO - ESTRUCTURA DE DATOS**



También podemos alternativamente usar esta declaración:

```
char arrayCaracteres[ ];  
arrayCaracteres = new char[10];  
tipo_dato nombre_array[ ]=new tipo_dato[tamaño];
```



*Java comprueba en tiempo de compilación que los índices estén dentro del rango, en caso contrario genera un error y durante la ejecución del programa un acceso fuera de rango genera una excepción.*



### Inicializar Arreglos

```
rectangulos[0]=new Rectangulo(10, 20, 30, 40);  
x= new int [100];
```

O bien, en una sola línea

```
Rectangulo[ ] rectangulos={new Rectangulo(10, 20, 30, 40),  
new Rectangulo(30, 40), new Rectangulo(50, 80)};
```

```
Int [ ] x = new int [100];
```

#### 1.6.1. Tamaño de los Arreglos, atributo length

- Java considera cada arreglo como un objeto.
- El número de elementos de un arreglo se conoce accediendo al campo length
- double [] a = new double [15]
- System.out.print (a.length); // escribe 15
- El campo length está protegido, no se puede modificar
- El número de elementos de un arreglo es un campo del mismo, no un método:
  - a.length; // correcto
  - a.length( ); // incorrecto

Esta variable nos devuelve el número de elementos que posee el array. Hay que tener en cuenta que es una variable de solo lectura, es por ello que no podremos realizar una asignación a dicha variable.



**PROGRAMA INGENIERIA DE SOFTWARE**  
**Modalidad de Educación a Distancia**  
**MODULO - ESTRUCTURA DE DATOS**



Por ejemplo esto nos serviría a la hora de mostrar el contenido de los elementos de un array:

```
char array[];  
array = new char[10];  
  
for (int x=0;x<array.length;x++)  
    System.out.println(array[x]);
```



*Uno de los axiomas de la orientación a objetos es la ocultación, es decir, que no podemos acceder a una variable declarada dentro de una clase a no ser que lo hagamos a través de un método de la clase. Aquí estamos accediendo a una variable. ¿Quizás sea porque no consideran a los arrays como objetos?*

**Ejemplo 1:**

```
public class Ejemplo1 {  
    public static void main(String[] args) {  
        int numeros[] = {10,15,3,80,65,74,1,65,35,44,9};  
        int min = numeros[0];  
        for(int i=1; i<numeros.length; i++) {  
            if(numeros[i] < min)  
                min = numeros[i];  
        }  
        System.out.println("El valor más pequeño es: "+min);  
    }  
}
```



Run:

El valor más pequeño es: 1



**PROGRAMA INGENIERIA DE SOFTWARE**  
**Modalidad de Educación a Distancia**  
**MODULO - ESTRUCTURA DE DATOS**



**Ejemplo 2:**

```
public class Ejemplo2 {
    public static void main(String[] args) {
        final int TAMANO = 26;
        char listaLetras[] = new char[TAMANO];
        char letra = 'A';
        //Asignacion de letras a cada elemento del vector
        for(int i=0; i<TAMANO; i++) {
            listaLetras[i] = letra;
            letra++;
        }
        //Mostrar en pantalla el vector
        for(int i=0; i<TAMANO; i++) {
            System.out.print(listaLetras[i]+" ");
        }
    }
}
```

Run:

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

**Ejemplo 3:**

```
public class Ejemplo3 {
    public static void main(String[] args) {
        int edades[] = {26,73,84,52,76,72,37,67,62,73};
        //Mostrar el primer elemento
        System.out.println(edades[0]);
        //Mostrar el último elemento
        System.out.println(edades[edades.length-1]);
        //Cambiar el tercer valor y mostrarlo
        edades[2] = 48;
        System.out.println(edades[2]);
        //Mostrar todo el vector
        for(int i=0; i<edades.length; i++)
            System.out.print(edades[i]+" ");
        System.out.println();
        //Mostrar todo el vector de otra forma
        for(int edad:edades)
```



**PROGRAMA INGENIERIA DE SOFTWARE**  
**Modalidad de Educación a Distancia**  
**MODULO - ESTRUCTURA DE DATOS**



```
        System.out.print(edad+" ");  
        System.out.println();  
    }  
}
```

Run:

```
26  
73  
48  
26 73 48 52 76 72 37 67 62 73  
26 73 48 52 76 72 37 67 62 73
```

**Ejemplo 4:**

```
public class Ejemplo4 {  
    public static void main(String[] args) {  
        int apuesta[] = {22,28,3,13,40,7};  
        int aciertos, bola, apuestas=0;  
        do {  
            aciertos = 0;  
            apuestas++;  
            for(int i=0;i<6;i++) {  
                bola = (int)(Math.random()*49)+1;  
                for(int j=0;j<6;j++)  
                {  
                    if(bola==apuesta[j])  
                        aciertos++;  
                }  
            }  
        } while(aciertos<6);  
        System.out.println("Apuestas realizadas: "+apuestas);  
    }  
}
```



Run:

Apuestas realizadas: 1085088

**Tarea:** modificar los programas a excepción del número dos, de tal manera que los vectores se han de 10 elementos y se introduzcan por teclado los valores.



**PROGRAMA INGENIERIA DE SOFTWARE**  
**Modalidad de Educación a Distancia**  
**MODULO - ESTRUCTURA DE DATOS**



### 1.6.2. Copiar y Comparar Arreglos en Java?

Muchas veces nos vamos a encontrar ante la disyuntiva de tener que copiar los elementos de un array en otros. Normalmente será para manipular el contenido del mismo guardando en uno de los arrays los datos originales.

Asumida ya la situación, lo primero que se nos ocurriría, independientemente del lenguaje en el que nos encontremos, será el montar un algoritmo que recorriendo el primer array vaya copiando dichos elementos en el segundo.

En Java nos quedaría un código como este:

```
for (int x=0;x<aOrigen.length;x++)  
    aDestino[x] = aOrigen[x];
```

En este sentido nada que objetar, ya que es muy buena práctica de programación. Pero todo buen programador tiene que tener en mente el concepto de "reutilización". No "reutilización" cómo copia (o fusilamiento -argot de muchos programadores-) del código. Sino "reutilización" pensando en que alguien ya puede haberse encontrado el problema y haberle dado ya una solución.

Para poder reutilizar tenemos que ser conscientes de lo que el entorno en el que estamos nos ofrece. Y en el caso de Java, es la librería del sistema la que nos ofrece una función para la copia de arrays. Como vemos en el siguiente código:

```
System.arrayCopy(aOrigen,inicioArrayOrigen, aDestino,  
    inicioArrayDestino,numeroElementosACopiar);
```

Al trabajar con arrays de tipos primitivos o de objetos se nos puede plantear la necesidad de copiar arrays. La copia de arrays está permitida pero conviene ser cauto cuando realicemos procesos de este tipo. Recordar que un array es un objeto (aunque sea especial) y por tanto la variable que lo nombra en realidad contiene un puntero al objeto, no el objeto en sí mismo.





**PROGRAMA INGENIERIA DE SOFTWARE**  
**Modalidad de Educación a Distancia**  
**MODULO - ESTRUCTURA DE DATOS**



Al hacer una operación del tipo `array1 = array2`, el puntero de `array1` apunta al mismo objeto que `array2` mientras que el objeto al que apuntaba `array1` queda inaccesible. A partir de ese momento existe identidad entre los arrays y la comparación usando `==` nos devolverá `true`. A través de código, vamos a plantearnos distintas situaciones y a comentar cuáles son los resultados.



*Hay que tener cuidado la función `arrayCopy` ya que esta nos puede devolver las siguientes excepciones: `IndexOutOfBoundsException` si intentamos copiar fuera del área reservado para el array, `ArrayStoreException` si intentamos copiar arrays de diferente tipo o `NullPointerException` si alguno de los array es nulo (vamos, no inicializado).*

**Ejemplo1.** Copiar estableciendo una relación de identidad entre arrays (aplicable a arrays de tipos primitivos y a arrays de objetos).

```
public class TestCopiaArrays {  
    public static void main (String [ ] Args) {  
        int [ ] miArray1 = {2, -4, 3, -7};  
        for (int i=0; i<miArray1.length; i++) {  
            System.out.print ("miArray1[" + i + "]= " + miArray1[i] + "; ");  
        }  
        System.out.println("");  
        int [ ] otroArray = {1, 2, 4, 8};  
        for (int i=0; i<otroArray.length; i++) {  
            System.out.print ("otroArray[" + i + "]= " + otroArray[i] + "; ");  
        }  
    }  
}
```





**PROGRAMA INGENIERIA DE SOFTWARE**  
**Modalidad de Educación a Distancia**  
**MODULO - ESTRUCTURA DE DATOS**



```
System.out.println("");
System.out.println ("¿Son el mismo objeto? ... " + (miArray1==otroArray) );
System.out.println("");
otroArray = miArray1; //otroArray pasa a ser el mismo objeto que miArray1
for (int i=0; i<otroArray.length; i++) {
    System.out.print ("otroArray[" + i +"]= " + otroArray[i]+"; ");
System.out.println("");
System.out.println ("¿Son el mismo objeto? ... " + (miArray1==otroArray) );
} //Cierre del main

} //Cierre de la clase
```

### **1.6.3. La clase arrays del api de java. Equals, copyof, fill.**

En la documentación de la clase Arrays del API de Java podemos encontrar, entre otras cosas, lo siguiente:

```
java.util
Class Arrays
java.lang.Object
|----- java.util.Arrays
```

Esta clase contiene varios métodos para manipular arrays (por ejemplo para ordenar un array o buscar un valor u objeto dentro de él) y para comparar arrays.



**PROGRAMA INGENIERIA DE SOFTWARE**  
**Modalidad de Educación a Distancia**  
**MODULO - ESTRUCTURA DE DATOS**



Dado que pertenece al package util, para poder usar esta clase habremos de incluir en cabecera import java.util.Arrays; o bien import java.util.\*;. Al igual que los arrays son unos objetos que hemos dicho son especiales (al carecer de métodos), podemos decir que la clase Arrays es una clase un tanto especial por cuanto carece de constructor. Digamos que directamente al cargar la clase con la sentencia import correspondiente automáticamente se crea un objeto denominado Arrays que nos permite realizar manipulaciones con uno o varios arrays (p. ej. ordenar un array, comparar dos arrays, etc.). Dicho objeto podemos utilizarlo directamente: no es necesario declararlo ni crearlo, eso es automático en Java, y por eso decimos que esta clase es una clase especial. La clase Arrays tiene muchos métodos, entre ellos varios métodos equals (sobrecarga del método) que hacen que equals sea aplicable tanto a arrays de los distintos tipos primitivos como a arrays de objetos. En concreto el método aplicable a arrays de enteros primitivos es:

static boolean **equals** (int[ ] a,int[ ] a2)

Devuelve true si los dos arrays especificados tienen relación de igualdad entre sí.

Vamos a aplicar este método para comparar el contenido de dos arrays de enteros (relación de igualdad). La aplicación al resto de tipos primitivos y objetos es análoga. La sintaxis en general es: Arrays.equals (nombreArray1, nombreArray2).



**PROGRAMA INGENIERIA DE SOFTWARE**  
**Modalidad de Educación a Distancia**  
**MODULO - ESTRUCTURA DE DATOS**



**Ejemplo 2.** Comparar arrays (relación de igualdad) usando la clase arrays

```
import java.util.Arrays;

public class TestCompararArrays {
    public static void main (String [ ] Args) {
        int [ ] miArray1 = {2, -4, 3, -7};
        for (int i=0; i<miArray1.length; i++) {
            System.out.print ("miArray1[" + i +"]= " + miArray1[i]+"; ");
            System.out.println ("");
        }
        int [ ] otroArray = {2, -4, 3, -7};
        for (int i=0; i<otroArray.length; i++) {
            System.out.print ("otroArray[" + i +"]= " + otroArray[i]+"; ");
            System.out.println ("¿Son el mismo objeto? ... " + (miArray1==otroArray) );
            System.out.println ("¿Tienen el mismo contenido (relación de igualdad)? ... " +
                Arrays.equals(miArray1, otroArray) );
            otroArray = miArray1; //otroArray pasa a ser el mismo objeto que miArray1
            for (int i=0; i<otroArray.length; i++) { System.out.print ("otroArray[" + i +"]= " +
                otroArray[i]+"; "); }
            System.out.println ("¿Son el mismo objeto? ... " + (miArray1==otroArray) );
            System.out.println ("¿Tienen el mismo contenido (relación de igualdad)? ... " +
                Arrays.equals(miArray1, otroArray) );
        } //Cierre del main
    } //Cierre de la clase
}
```



El resultado es correcto, porque hemos usado correctamente la clase Arrays para realizar la comparación entre dos arrays.



**PROGRAMA INGENIERIA DE SOFTWARE**  
**Modalidad de Educación a Distancia**  
**MODULO - ESTRUCTURA DE DATOS**



**Ejemplo 3.** Copiar contenidos entre arrays sin establecer relación de identidad ( “manual”, aplicable a tipos primitivos y a objetos).

Antes vimos cómo asignar el contenido de un array a otro haciendo que la variable apunte al mismo objeto. Vamos a ver ahora cómo copiar el contenido entre dos arrays pero manteniendo que cada variable denominadora del array apunte a un objeto diferente:

```
import java.util.Arrays;

//Test copia arrays con igualdad sin identidad aprenderaprogramar.com
public class TestCopiaConIgualdadSinIdentidad {
    public static void main (String [ ] Args) {
        int [ ] miArray1 = {2, -4, 3, -7};
        for (int i=0; i<miArray1.length; i++) {
            System.out.print ("miArray1[" + i +"]= " + miArray1[i]+"; ");
            System.out.println();
        }
        int [ ] otroArray = {1, 2, 4, 8};
        for (int i=0; i<otroArray.length; i++) {
            System.out.print ("otroArray[" + i +"]= " + otroArray[i]+"; ");
            System.out.println ("¿ Son el mismo objeto? ... " + (miArray1==otroArray)
                );
            System.out.println ("¿ Tienen el mismo contenido (relación de igualdad)? ...
                " + Arrays.equals(miArray1, otroArray) );
        }
        //Realizamos una asignación elemento a elemento
        for (int i=0; i < otroArray.length; i++) {
            otroArray[i] = miArray1[i];
        }
        for (int i=0; i < otroArray.length; i++) {
```



**PROGRAMA INGENIERIA DE SOFTWARE**  
**Modalidad de Educación a Distancia**  
**MODULO - ESTRUCTURA DE DATOS**



```
System.out.print ("otroArray[" + i +"]= " + otroArray[i]+"; ");}

System.out.println ("¿Son el mismo objeto? ... " + (miArray1==otroArray) );
System.out.println ("¿Tienen el mismo contenido (relación de igualdad)? ... "
+ Arrays.equals(miArray1, otroArray) );
} //Cierre del main
} //Cierre de la clase
```

**Ejemplo 4.** Copiar contenidos entre arrays sin establecer relación de identidad (Usando el método copyOf de la clase Arrays, aplicable a tipos primitivos y a objetos).

El método copyOf de la clase Arrays nos permite:

- a) Copiar un array manteniendo el número de elementos.
- b) Copiar un array agrandando el número de elementos que tiene, quedando los nuevos elementos rellenos con valores cero o nulos.
- c) Copiar un array empequeñeciendo el número de elementos que tiene; los elementos que no caben en el nuevo array, dado que tiene menor capacidad, se pierden (el array queda truncado).

copyOf es un método sobrecargado. En el caso de arrays de enteros su signatura es la siguiente:

```
static int[ ] copyOf (int[ ] original, int newLength)
```

Copia el array especificado, truncando o relleno con ceros (si fuera necesario) de manera que la copia tenga el tamaño especificado.



**PROGRAMA INGENIERIA DE SOFTWARE**  
**Modalidad de Educación a Distancia**  
**MODULO - ESTRUCTURA DE DATOS**



Para el resto de tipos primitivos su sintaxis es análoga: `Arrays.copyOf (nombreDelArray, n)` siendo `n` un entero que define la nueva longitud del array (`n` puede ser mayor, menor o igual que la longitud del array original). El código de ejemplo sería este (usamos el `copyOf` sin variar la longitud del array):

```
import java.util.Arrays;

//Test uso de copyOf método clase Arrays aprenderaprogramar.com
public class TestUso_copyOf_1 {
    public static void main (String [ ] Args) {
        int [ ] miArray1 = { 2, -4, 3, -7 };
        for (int i=0; i<miArray1.length; i++) {
            System.out.print ("miArray1[" + i +"]= " + miArray1[i]+" ");
        }
        System.out.println ("");
        int [ ] otroArray = { 1, 2, 4, 8 };
        for (int i=0; i<otroArray.length; i++) {
            System.out.print ("otroArray[" + i +"]= " + otroArray[i]+" ");
        }
        System.out.println ("¿Son el mismo objeto? ... " + (miArray1==otroArray) );
        System.out.println ("¿Tienen el mismo contenido (relación de igualdad)? ... " +
Arrays.equals(miArray1, otroArray) );

        //Copiamos el array utilizando el método copyOf de la clase Arrays
        otroArray = Arrays.copyOf(miArray1, miArray1.length);

        for (int i=0; i<otroArray.length; i++) {
            System.out.print ("otroArray[" + i +"]= " + otroArray[i]+" ");
        }
    }
}
```



**PROGRAMA INGENIERIA DE SOFTWARE**  
**Modalidad de Educación a Distancia**  
**MODULO - ESTRUCTURA DE DATOS**



```
System.out.println ("¿Son el mismo objeto? ... " + (miArray1==otroArray) );  
System.out.println ("¿Tienen el mismo contenido (relación de igualdad)? ... " +  
Arrays.equals(miArray1, otroArray) );  
    } //Cierre del main  
} //Cierre de la clase
```

Hemos comprobado que **el método copyOf de la clase Arrays realiza una copia elemento a elemento** entre los contenidos de dos arrays pero no hace que los punteros apunten al mismo objeto. Prueba a variar la longitud que se le pasa como parámetro al método copyOf, por ejemplo:

```
otroArray = Arrays.copyOf(miArray1, miArray1.length+2); //Resto del código igual  
otroArray = Arrays.copyOf(miArray1, miArray1.length-2); //Resto del código igual
```

Comprueba que los resultados son el alargamiento del array y su relleno con ceros, o el acortamiento con pérdida de los datos (truncamiento) que no caben debido al recorte de la longitud. En el caso de alargamiento o expansión del array cuando se trata de un array que no sea de enteros, si son tipos numéricos se rellenan los excedentes con ceros, si son booleanos se rellenan los excedentes con false, si son char se rellenan de caracteres vacío, y si son objeto se rellenan los excedentes con null.

#### **1.6.4. Rellenar un array con un valor u objeto. Método fill de la clase arrays**

La clase Arrays tiene un método, denominado fill, sobrecargado, que permite rellenar un array con un determinado valor u objeto. En el caso de arrays de enteros la signatura es:

```
static void fill (int[ ] a, int val)
```



**PROGRAMA INGENIERIA DE SOFTWARE**  
**Modalidad de Educación a Distancia**  
**MODULO - ESTRUCTURA DE DATOS**



Asigna el valor entero especificado a cada elemento del array de enteros indicado.

En general la sintaxis será: Arrays.fill (nombreDelArray, valor con el que se rellena). El valor con el que se rellena depende del tipo del array. Por ejemplo, si es un array de tipo booleano, tendremos que rellenarlo bien con true o bien con false, no podremos rellenarlo con un tipo que no sea coherente. Ejemplos de uso:

Arrays.fill (resultado, '9'); Como rellenamos con un carácter, resultado habrá de ser un array de caracteres, ya que en caso contrario no habría coincidencia de tipos.

Arrays.fill (permitido, true); Como rellenamos con un true, resultado será un array de booleanos. De otra manera, no habría coincidencia de tipos. Ejemplo de código:

```
import java.util.Arrays;
public class TestMetodoFillArrays {
    public static void main (String [ ] Args) { //main cuerpo del programa ejemplo
aprenderaprogramar.com
        int [ ] miArray = new int[10];
        Arrays.fill(miArray, 33);
        for (int tmp: miArray) { System.out.print (tmp + ","); } //Recorrido del array con un
for each
    } } //Cierre del main y de la clase
```



En caso de que el array tenga contenidos previos al aplicarle el fill, todos sus elementos quedarán reemplazados por el elemento de relleno. No obstante, hay otro método que permite especificar los índices de relleno de modo que se pueda preservar parte del contenido previo del array:

```
static void fill (int[ ] a, int fromIndex, int toIndex, int val)
```





**PROGRAMA INGENIERIA DE SOFTWARE**  
**Modalidad de Educación a Distancia**  
**MODULO - ESTRUCTURA DE DATOS**



Asigna el valor entero especificado a cada elemento del rango indicado para el array especificado.

Escribe un fragmento de código utilizando esta signatura del método fill y comprueba sus resultados.

## **EJERCICIO**

Crea un programa Java donde declares un array de enteros tipo int miArray1 cuyo contenido inicial sea {2, -4, 3, -7}. Muestra su contenido por pantalla. Copia el contenido de este array a un ArrayList denominado lista1 y muestra su contenido por pantalla. ¿Qué tipo de datos almacena el array? ¿Qué tipo de datos almacena el ArrayList?

### **1.6.5. Recorrer un Array Unidimensional**

Para recorrer un array se utiliza una instrucción iterativa (normalmente una instrucción for, aunque también puede hacerse con while o do..while) utilizando una variable entera como índice que tomará valores desde el primer elemento al último o desde el último al primero.

Por ejemplo, el siguiente fragmento de programa Java declara un array de 7 elementos de tipo double y le asigna valores iniciales. A continuación recorre el array, utilizando la instrucción for, para mostrar por pantalla el contenido del array.

```
double[] notas = {2.3, 8.5, 3.2, 9.5, 4, 5.5, 7.0}; //array de 7 elementos
for (int i = 0; i < 7; i++) {
```



**PROGRAMA INGENIERIA DE SOFTWARE**  
**Modalidad de Educación a Distancia**  
**MODULO - ESTRUCTURA DE DATOS**



```
System.out.print(notas[i] + " "); //se muestra cada elemento del array  
}
```

Para evitar errores de acceso al array es recomendable utilizar `length` para recorrer el array completo.

Por ejemplo:

```
double[] notas = {2.3, 8.5, 3.2, 9.5, 4, 5.5, 7.0}; //array de 7 elementos  
for (int i = 0; i < notas.length; i++) {  
    System.out.print(notas[i] + " "); //se muestra cada elemento del array  
}
```

**Ejemplo de recorrido de un array en java:**

Programa que lee por teclado la nota de los alumnos de una clase y calcula la nota media del grupo. También muestra los alumnos con notas superiores a la media. El número de alumnos se lee por teclado.

Este programa crea un array de elementos de tipo `double` que contendrá las notas de los alumnos. El tamaño del array será el número de alumnos de la clase.

Se realizan **3 recorridos** sobre el array, el primero para asignar a cada elemento las notas introducidas por teclado, el segundo para sumarlas y el tercero para mostrar los alumnos con notas superiores a la media.



**PROGRAMA INGENIERIA DE SOFTWARE**  
**Modalidad de Educación a Distancia**  
**MODULO - ESTRUCTURA DE DATOS**



```
import java.util.*;

public class Recorrido2 {

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int numAlum, i;
        double suma = 0, media;
        do {
            System.out.print("Número de alumnos de la clase: ");
            numAlum = sc.nextInt();
        } while (numAlum <= 0);
        double[] notas = new double[numAlum]; //se crea el array
        // Entrada de datos. Se asigna a cada elemento del array
        // la nota introducida por teclado
        for (i = 0; i < notas.length; i++) {
            System.out.print("Alumno " + (i + 1) + " Nota final: ");
            notas[i] = sc.nextDouble();
        }
        // Sumar todas las notas
        for (i = 0; i < notas.length; i++) {
            suma = suma + notas[i];
        }
        // Calcular la media
        media = suma / notas.length;
        // Mostrar la media
        System.out.printf("Nota media del curso: %.2f %n", media);
        // Mostrar los valores superiores a la media
        System.out.println("Listado de notas superiores a la media: ");
```



**PROGRAMA INGENIERIA DE SOFTWARE**  
**Modalidad de Educación a Distancia**  
**MODULO - ESTRUCTURA DE DATOS**



```
for (i = 0; i < notas.length; i++) {  
    if (notas[i] > media) {  
        System.out.println("Alumno numero " + (i + 1)+ " Nota final: " + notas[i]);  
    }  
}  
}
```

#### **1.6.6. Recorrer un Array en java con for-each. Bucle for para colecciones**

A partir de java 5 se incorpora una instrucción for mejorada para recorrer arrays y contenedores en general.

Permite acceder secuencialmente a cada elemento del array.

La sintaxis general es:

```
for(tipo nombreDeVariable : nombreArray){  
    ....  
}
```

**tipo:** indica el tipo de datos que contiene el array.

**nombreDeVariable:** variable a la que en cada iteración se le asigna el valor de cada elemento del array. Está definida dentro del for por lo que solo es accesible dentro de él.



**PROGRAMA INGENIERIA DE SOFTWARE**  
**Modalidad de Educación a Distancia**  
**MODULO - ESTRUCTURA DE DATOS**



nombreArray: es el nombre del array que vamos a recorrer.

Mediante este bucle solo podemos acceder a los elementos del array. No podemos hacer modificaciones en su contenido.

Ejemplo: El siguiente programa crea un array temperatura de 10 elementos. Lee por teclado los valores y a continuación los muestra por pantalla.

```
import java.util.*;

public class Recorrerforeach1 {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        double [] temperatura = new double[10];
        int i;
        for(i = 0; i<temperatura.length;i++){
            System.out.print("Elemento " + i + ": ");
            temperatura[i] = sc.nextDouble();
        }

        for(double t: temperatura){
            System.out.print(t + " ");
        }
        System.out.println();
    }
}
```



**PROGRAMA INGENIERIA DE SOFTWARE**  
**Modalidad de Educación a Distancia**  
**MODULO - ESTRUCTURA DE DATOS**



**1.6.7. ¿Por qué se clasifica un arreglo?**

Esto se hace por lo general para facilitar la búsqueda de los elementos del array. Así se clasifica en: los diccionarios, las agendas telefónicas, los casilleros de bibliotecas, relación de amigos, etc.

Podemos clasificar a las operaciones que intervienen arreglos de la siguiente manera:

- Lectura/Escritura
- Asignación
- Actualización: Inserción, Eliminación, Modificación
- Ordenación
- Búsqueda

Como los arreglos son datos estructurados, muchas de estas operaciones no pueden llevarse a cabo de manera global, sino que se debe trabajar sobre cada componente.



# PROGRAMA INGENIERIA DE SOFTWARE

## Modalidad de Educación a Distancia

### MODULO - ESTRUCTURA DE DATOS



#### Lectura de un Arreglo:

#### LA LECTURA DE LA CONSOLA: JAVA SCANNER VS BUFFEREDREADER

Cuando se lee una entrada desde la consola, los dos métodos más comunes de la lectura de los datos basados en caracteres de un archivo en Java son en primer lugar el uso del Scanner, y en segundo lugar se usa BufferedReader. Ambos tienen características diferentes. Significa diferencias del cómo usarlo.

DIFERENCIAS	
Scanner	BufferedReader
Scanner tratado entrada dada como testigo.	BufferedReader acaba de leer línea por línea dada de entrada como cadena. BufferedReader como línea de corriente / Cadena
Scanner por sí mismo proporciona analizar capacidades como nextInt (), nextFloat ().	Un BufferedReader es una clase simple significado de leer de manera eficiente de la corriente de subalterno. Generalmente, cada solicitud de lectura hecha de un lector como un FileReader causa una solicitud de lectura correspondientes que deben introducirse en la corriente subyacente. Cada invocación de read () o readLine () podría causar bytes a leer desde el archivo, convertidos en personajes, y luego regresó, lo que puede ser muy ineficiente. La eficiencia se mejoró sensiblemente si un lector está deformado en un BufferedReader.
Un escáner por otro lado tiene mucho más queso construido en él; puede hacer todo lo que un BufferedReader puede hacer y al mismo nivel de eficiencia también. Sin embargo, además de un escáner puede analizar la secuencia subyacente para los tipos y cadenas primitivos utilizando expresiones regulares. También puede tokenize la corriente subyacente con el delimitador de su elección. También puede hacer exploración en avance de la corriente subyacente sin tener en cuenta el delimitador!	Usando BufferedReader debe escribir código adicional y sólo puede leer y almacenar String.
El escáner no está sincronizado. Un escáner no se hilo de seguridad, tiene que ser sincronizado externamente.	BufferedReader está sincronizado, por lo que las operaciones de lectura en un BufferedReader con seguridad se puede hacer desde varios subprocesos.
Scanner vienen con JDK desde la versión 1.5 más alto.	

¿Cuándo se debe utilizar un escáner o Buffered Reader?

La elección de utilizar un BufferedReader o un escáner depende del código que está escribiendo, si usted está escribiendo un lector de registro sencillo búfer es adecuada. Sin embargo, si usted está escribiendo un escáner analizador XML es la elección más natural.

Incluso durante la lectura de la entrada, si quieren aceptar la línea de entrada del usuario en línea y decir sólo tiene que añadir a un archivo, un BufferedReader es lo suficientemente bueno. Por otro lado, si desea aceptar la entrada del usuario como un comando con múltiples opciones, y luego la intención de realizar diferentes operaciones en base a la orden y opciones especificado, un escáner se adapte mejor.





**PROGRAMA INGENIERIA DE SOFTWARE**  
**Modalidad de Educación a Distancia**  
**MODULO - ESTRUCTURA DE DATOS**



**1.6.8. Utilización de dispositivos de almacenamiento externo.**



Con frecuencia las aplicaciones informáticas trabajan sobre datos almacenados de forma estable en dispositivos de almacenamiento (típicamente discos magnéticos).

Una primera vez el usuario crea un “**Archivo**” (*file*) y lo “guarda” en un disco. Posteriormente podrá utilizar dicho Archivo, consultar y modificar los datos, así como añadir nuevos o eliminar alguno/s de los actuales.

La utilización de Archivos en disco es muy amplia y va más allá de los objetivos de este curso. Simplemente indicar que los Archivos están constituidos por un conjunto de **registros** que definen las características de cada uno de los elementos (“Archivos”) del Archivo<sup>9</sup>.

Otro aspecto a considerar es cómo organizar las “Archivos” dentro del dispositivo (Archivo). Con soportes tradicionales la forma de organización es **única** (por ejemplo, alfabética) y **secuencial** (una ficha tras otra, en el orden predefinido). La Informática ha heredado esta idea aunque permite otras modalidades de organización y modos de acceso adicionales a los puramente únicos y secuenciales.

**1.6.8.1. Archivos de texto.**

En un Archivo de texto, cada una de sus “Archivos” consiste en una cadena de caracteres de longitud variable (líneas de texto). Los “Archivos” se almacenan una tras otra sin mantener ningún criterio de orden entre sí. Por ejemplo (*marsell.txt*):

Allons enfants de la  
Patrie, Le jour de  
gloire est arrivé!  
Contre nous de la  
tyrannie,  
L'étendard sanglant est  
levé,(bis) Entendez-vous



<sup>9</sup> Este concepto está heredado de los antiguos “Archivos” (o archivadores) que contenían “Archivos” (de papel) cada una de las cuales representaba una información unitaria: Archivos de películas, personas, asignaturas....



**PROGRAMA INGENIERIA DE SOFTWARE**  
**Modalidad de Educación a Distancia**  
**MODULO - ESTRUCTURA DE DATOS**



dans le campagnes, Mugir  
ces féroces soldats?  
Ils viennent jusque dans  
nos bras, Égorger nos fils,  
nos compagnes!

Entre cada una de las líneas existen “códigos invisibles” cuyo efecto es hacer saltar al inicio de la línea siguiente. Estos códigos se generan automáticamente al pulsar la tecla “Intro”. Así mismo, para indicar el final del Archivo de texto, el usuario deberá haber pulsado la tecla de función “F6”.

La figura siguiente muestra el código ASCII correspondiente al ejemplo.

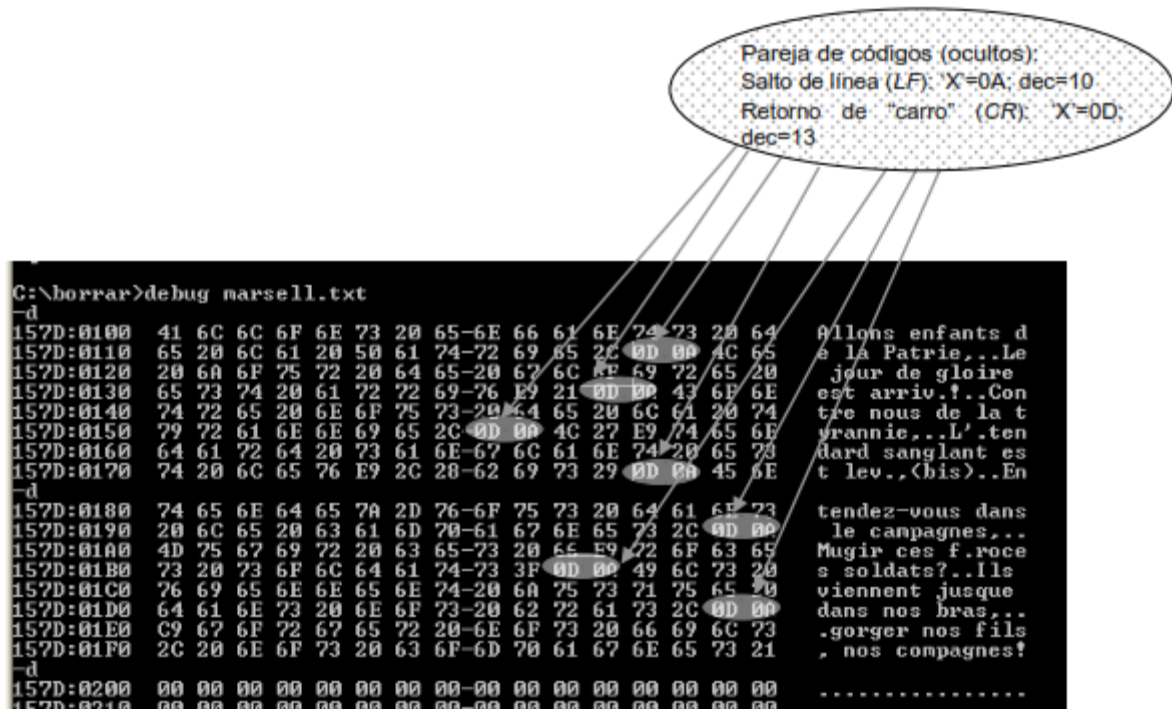


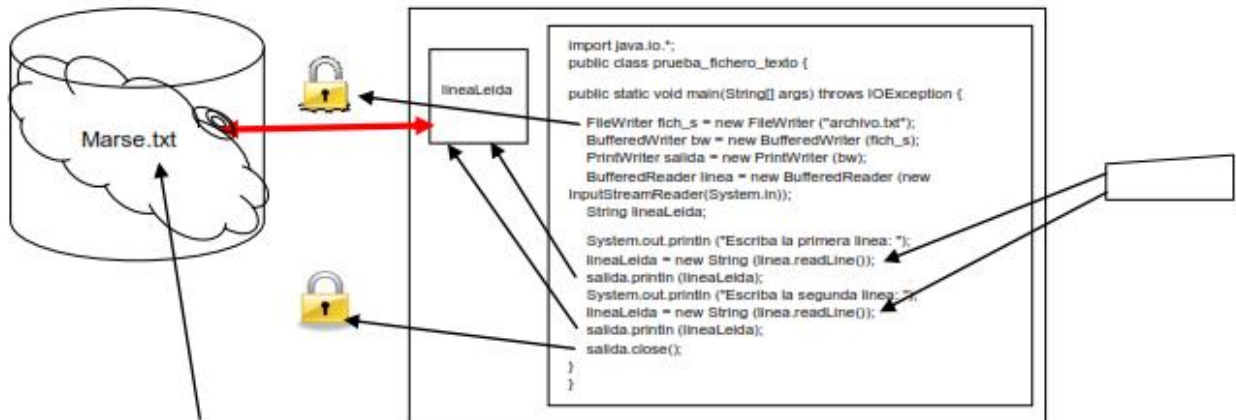
Figura 1.7. Códigos de fin de línea y retorno de carro en un Archivo de texto.



**PROGRAMA INGENIERIA DE SOFTWARE**  
**Modalidad de Educación a Distancia**  
**MODULO - ESTRUCTURA DE DATOS**



La figura siguiente muestra, a título de ejemplo, un programa que **crea** un Archivo (*archivo.txt*)<sup>10</sup> e introduce en él dos líneas de texto (procedentes del teclado y recogidas, sucesivamente, en la variable, de tipo **String**, *lineaLeida*). Una vez finalizado el proceso el Archivo deberá “cerrarse” para permitir posteriores usos del mismo.



*Figura 1.8. Funcionamiento del programa*

```
import java.io.*;
public class PruebaArchivoSalida {
    public static void main (String [ ] args) throws IOException {
        FileWriter fich_s = new FileWriter ("archivo.txt");
        BufferedWriter bw = new BufferedWriter (fich_s);
        PrintWriter salida = new PrintWriter (bw);
        BufferedReader linea = new BufferedReader (new InputStreamReader(System.in));
        String lineaLeida;

        System.out.println ("Escriba la primera linea: ");
        lineaLeida = new String (linea.readLine ());
        salida.println (lineaLeida);

        System.out.println ("Escriba la segunda linea: ");
        lineaLeida = new String (linea.readLine ());
        salida.println (lineaLeida);
        salida.close();
    }
}
```



<sup>10</sup> La ruta, nombre y extensión del Archivo se eligen libremente (con las restricciones propias del sistema operativo). En el ejemplo se han utilizado:

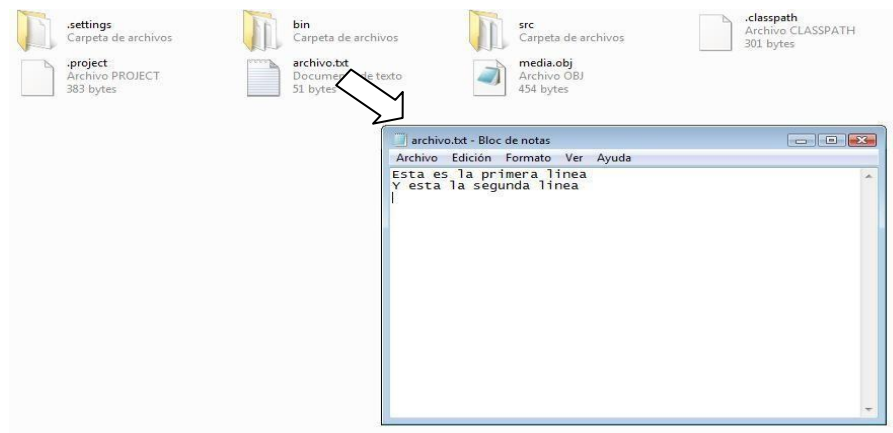
- Ruta: (por omisión) la misma que el programa. Nombre: *archivo*.
- Extensión *txt* (para poder visualizarlo mediante el *bloc de notas –notepad–* de Windows).

**PROGRAMA INGENIERIA DE SOFTWARE**  
**Modalidad de Educación a Distancia**  
**MODULO - ESTRUCTURA DE DATOS**



```
System.out.println ("Escriba la segunda linea: ");
lineaLeida = new String (linea.readLine ());
salida.println (lineaLeida);
salida.close ();
}
}
```

La figura siguiente ilustra un ejemplo de posible resultado de la ejecución del programa anterior.



*Figura 1.9. Resultado de la ejecución del programa*

#### 1.6.8.2. Principales consideraciones semánticas.

- El programador no es consciente del Archivo físico con que está trabajando. Se crea un nombre simbólico (por ejemplo *fich\_s*), y sobre él se realizan todas las operaciones<sup>11</sup>. Sólo existe una excepción, la sentencia:  
`FileWriter fich_s = new FileWriter ("archivo.txt");`  
establece una vinculación entre el nombre físico del Archivo y el nombre lógico.
- Se utiliza una variable de tipo ***String*** (en el ejemplo: *lineaLeida*) como “puente” entre la memoria del computador y el dispositivo de almacenamiento (disco). Según sea el sentido de la transferencia se puede hablar de lectura: disco → memoria (***readline***) o de escritura memoria → disco (***println***).

<sup>11</sup> Dicho Archivo estará gestionado por el sistema operativo. Con frecuencia se cambia de ubicación los Archivos (físicos) y de no seguir esta filosofía, esto implicaría re-escribir parte del código cada vez que un Archivo cambiase de ubicación.

**PROGRAMA INGENIERIA DE SOFTWARE**  
**Modalidad de Educación a Distancia**  
**MODULO - ESTRUCTURA DE DATOS**



- Hay que preparar el Archivo en disco para utilizarlo (**apertura**) y dejarlo disponible para posteriores usos (**cierre**).

**1.6.8.3. Sintaxis de las principales operaciones relacionadas con los Archivos de texto<sup>12</sup>.**

- Acceso (Asignación y apertura) y declaración de variables:
  - o Crear un nuevo Archivo y **destruir**, en su caso, un Archivo previo.

Posteriormente se podrá *escribir* en él:

```
FileWriter <nombre lógico> = new FileWriter (<nombre físico>);  
BufferedWriter <buffer escritura> = new BufferedWriter (<nombre  
lógico>); PrintWriter <variable> = new PrintWriter (<buffer escritura>);
```

- o Preparar un Archivo (que debe existir previamente) para poder *leer* posteriormente su contenido:

```
FileReader <nombre lógico> = new FileReader (<nombre físico>);  
BufferedReader <buffer lectura> = new BufferedReader (<nombre  
lógico>);
```

- Escribir en un Archivo (previamente creado) nuevas líneas de texto a partir de la última:

```
FileWriter <nombre lógico> = new FileWriter (<nombre físico>,true);
```

- Proceso:
  - *Leer* (transferir a la variable correspondiente) el contenido de una línea del Archivo y prepararse para la siguiente.  
`<variable tipo String> = <buffer lectura>.readline();`
  - *Escribir* (transferir) el contenido de una línea (`<expresión tipo String>`) al Archivo y prepararse para la siguiente:

```
<variable>.println = <expresión tipo String>;
```

- Terminación:
  - o Dejar el Archivo preparado para usos posteriores:  
`<variable>.close ();`

<sup>12</sup> Para mayor información consultar el manual de programación



**PROGRAMA INGENIERIA DE SOFTWARE**  
**Modalidad de Educación a Distancia**  
**MODULO - ESTRUCTURA DE DATOS**



Por ejemplo, si queremos leer un Archivo hasta llegar al final lo podemos hacer utilizando lo siguiente:

```
import java.io.*;
public class PruebaArchivoEntrada {
    public static void main (String [ ] args) throws IOException { String
        lineaLeida;
        FileReader fichLeido = new FileReader ("archivo.txt");
        BufferedReader entrada = new BufferedReader (fichLeido);

        System.out.println ("Contenido del Archivo: ");
        while ((lineaLeida = entrada.readLine ()) != null)
            System.out.println (lineaLeida);
        entrada.close ();
    }
}
```



#### 1.6.8.4. Archivos binarios.

En un Archivo binario, las “Archivos” son elementos de tipo **registro**. En consecuencia (entre otras) no se puede visualizar su contenido con un visor de texto del sistema operativo (por ejemplo, el bloc de notas de Windows).

Para poder guardar objetos (por ejemplo, del tipo *RegistroAlumno* visto en apartados anteriores) en un Archivo hay que hacerlos “serializables”. Mediante la serialización, un objeto se convierte en una secuencia de bytes con la que se puede reconstruir posteriormente manteniendo el valor de sus variables. Esto permite guardar un objeto en un archivo o mandarlo por red. Una clase se serializa añadiendo en su definición:

implements Serializable

Para poder leer y escribir objetos que se han declarado como *serializables* se utilizan las clases *ObjectInputStream* y *ObjectOutputStream*, que cuentan con los métodos *writeObject()* y *readObject()*.

Para escribir un objeto en un Archivo se utilizará:





**PROGRAMA INGENIERIA DE SOFTWARE**  
**Modalidad de Educación a Distancia**  
**MODULO - ESTRUCTURA DE DATOS**



```
ObjectOutputStream <objEscrito> = new ObjectOutputStream (new  
    FileOutputStream("<nombre Archivo>"));  
<objEscrito>.writeObject (<variable>);
```

Mientras que las instrucciones utilizadas para leerlos después serían:

```
ObjectInputStream <objLeido> = new ObjectInputStream (new  
    FileInputStream("<nombre Archivo>"));  
<tipo> <variable> = (<tipo>) <objLeido>.readObject ();
```

Cuando se ha terminado de leer o escribir el Archivo correspondiente es necesario cerrar el Archivo con:

```
<nombre Archivo>.close()
```

El siguiente ejemplo es una variante del programa de gestión de alumnos visto en el apartado 1.4.2.2. Sus características más importantes son<sup>13</sup>:

- Concepción modular: un programa principal y un conjunto de subprogramas.
- No se utilizan variables globales.

El programa principal:

- o Prepara el Archivo en disco:

```
String nombre;  
BufferedReader linea = new BufferedReader (new  
    InputStreamReader(System.in));
```

```
System.out.println ("Introducir nombre del Archivo: ");  
nombre = new String (linea.readLine());
```

- o Su lógica se basa en un planteamiento de “tipo menú”:
  - Llama al módulo que realiza la interfaz de usuario: *menu* (sin argumentos)

<sup>13</sup> Por simplicidad no se han considerado situaciones excepcionales (“por excepción”). Por ejemplo: No tiene sentido utilizar la opción [2]: “Cargar tabla de registros”, si no existe el Archivo en disco. Tampoco lo tiene utilizar la opción [3]: “Calcular calificación media”, si no se ha ejecutado (con éxito) previamente la opción [2].





**PROGRAMA INGENIERIA DE SOFTWARE**  
**Modalidad de Educación a Distancia**  
**MODULO - ESTRUCTURA DE DATOS**



- Recibe (variable *op*) la opción seleccionada, la analiza (mediante una estructura **case**) e invoca al módulo correspondiente:
    - *crearArchivo* (*Archivo*).
    - *cargarTabla* (*alumnos*, *Archivo*)
    - *mediaCalif* (*alumnos*)
  - o Utiliza el menor conjunto posible de información (variables). Obsérvese que el programa principal no necesita para nada utilizar registros correspondientes a alumnos individuales (tipo *RegistroAlumno*).
  - El procedimiento *menu*. Es autónomo (no necesita recibir ni devolver argumentos).
  - La función *mediaCalif* (*alumnos*). El único argumento que necesita es una tabla de registros de alumnos (*RegistroAlumno []*).
  - El procedimiento *cargarTabla* (*alumnos*, *Archivo*). Utiliza como información de entrada un *Archivo* y genera en memoria (*alumnos*) una estructura *RegistroAlumno []* que quedará a disposición del programa principal. Se encarga de abrir, en modo lectura, el *Archivo* y cerrarlo al finalizar.
  - El procedimiento *crearArchivo* (*Archivo*). Genera un *Archivo* binario en disco a partir de los datos introducidos por el usuario desde el teclado. Abre el *Archivo* en modo escritura y lo cierra al terminar.
- El siguiente esquema muestra gráficamente la arquitectura de la aplicación.

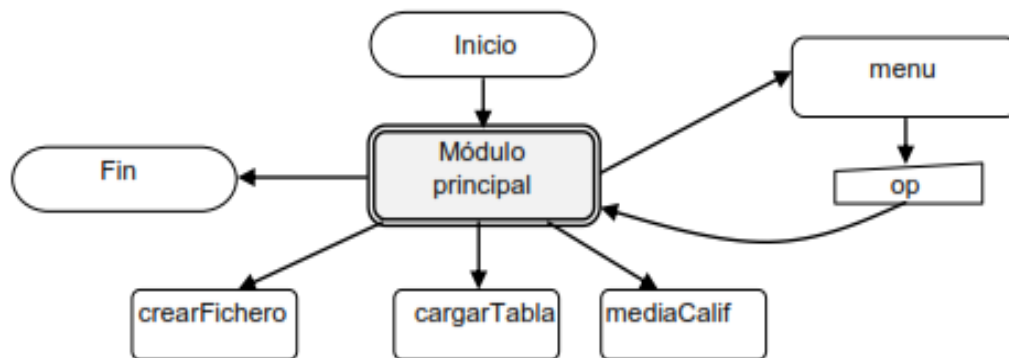


Figura 1.10. Arquitectura del programa ejemplo.



**PROGRAMA INGENIERIA DE SOFTWARE**  
**Modalidad de Educación a Distancia**  
**MODULO - ESTRUCTURA DE DATOS**



El código del ejemplo sería:

```
import java.io.*;

public class PruebaFichES {

    static void crearArchivo (RegistroAlumno [ ] alumnos, String nombref) throws
    IOException {
        int i;
        FileOutputStream fich = new FileOutputStream (nombref);
        ObjectOutputStream objEscrito = new ObjectOutputStream
        (fich);
        for (i = 0; i < 6; i++) {
            alumnos [i] = new RegistroAlumno();
            System.out.println ("Datos del alumno N: "+ i);
            alumnos [i].cargarRegistro ();
            objEscrito.writeObject (alumnos [i]);
        }
        fich.close ();
    }

    static void cargarTabla (RegistroAlumno [ ] alumnos, String nombref) throws
    IOException, ClassNotFoundException {
        int i;
        FileInputStream fich =new FileInputStream (nombref);
        ObjectInputStream objLeido = new ObjectInputStream (fich);

        for (i = 0; i < 6; i++) {
            alumnos [i] = (RegistroAlumno) objLeido.readObject ();
            System.out.println ("Datos del alumno N: " + i);
            System.out.println(alumnos [i].aCadena ());
        }
        fich.close ();
    }
}
```



**PROGRAMA INGENIERIA DE SOFTWARE**  
**Modalidad de Educación a Distancia**  
**MODULO - ESTRUCTURA DE DATOS**



```
static float mediaCalif (RegistroAlumno [ ] alumnos) {  
    float resul;  
    int i;  
  
    resul = 0;  
    for (i = 0; i < 6; i++) {  
        System.out.println (alumnos [i].aCadena ());  
        resul = resul + alumnos [i].calificacion;} return  
        resul/6;  
    }  
}
```



```
static void menu () { System.out.println  
    ("OPCIONES:"); System.out.println ("Opcion 1:  
    Crear Archivo.");  
    System.out.println ("Opcion 2: Cargar tabla de registros.");  
    System.out.println ("Opcion 3: Calcular calificacion media.");  
    System.out.println ("Opcion 0: Salir.");  
    System.out.println ("\n Introduzca opcion: ");  
}
```

```
public static void main (String[] args) throws IOException,  
ClassNotFoundException {
```

```
    RegistroAlumno [ ] alumnos = new RegistroAlumno [6];  
    float media;  
    int op;  
    String nombre;  
    BufferedReader linea = new BufferedReader (new InputStreamReader(System.in));  
  
    System.out.println ("Introducir nombre del Archivo: ");  
    nombre = new String (linea.readLine ());  
    menu();  
    op = Integer.parseInt (linea.readLine ());  
    while (op != 0) {  
        switch (op) {  
            case 1: crearArchivo (alumnos,nombre);  
                    break;  
            case 2: cargarTabla (alumnos, nombre);  
                    break;  
            case 3: media = mediaCalif (alumnos);
```



**PROGRAMA INGENIERIA DE SOFTWARE**  
**Modalidad de Educación a Distancia**  
**MODULO - ESTRUCTURA DE DATOS**



```
        System.out.println ("La media de las calificaciones es: "+media);
        break;
    default: System.out.println("opción no valida");
        break;
    }
    menu ();
    op = Integer.parseInt (linea.readLine ());
}
System.out.println ("Adios");
}
}
```

● **Escritura de un Arreglo:**

**1.6.9. ¿Por qué se ordena un arreglo?**

Ordenar un array es muy importante, ya sea de números o de cadenas, puede haber casos en que nos interese que los datos estén ordenados en un array.

**1.6.9.1. Ordenamiento por el Método de Burbuja**

Es uno de los métodos de ordenación más conocidos y uno de los primeros que aprenden los programadores.

Consiste en comparar pares de elementos adyacentes en un array y si están desordenados intercambiarlos hasta que estén todos ordenados.

Si A es el array a ordenar, se realizan A.length-1 pasadas. Si la variable i es la que cuenta el número de pasadas, en cada pasada i se comprueban los elementos



**PROGRAMA INGENIERIA DE SOFTWARE**  
**Modalidad de Educación a Distancia**  
**MODULO - ESTRUCTURA DE DATOS**



adyacentes desde el primero hasta A.length-i-1 ya que el resto hasta el final del array están ya ordenados. Si los elementos adyacentes están desordenados se intercambian.

Ejemplo:

```
/*
 * Realizar un programa que solicita al usuario una lista de números por teclado y las
 * ordena con el Método Burbuja (Descendentemente)
 */
package ejemplo6;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;

/**
 *
 * @author Astrid
 */
public class Ejemplo6 {

    /**
     * @param arg
     * @throws java.io.IOException
     */
    public static void main(String arg[]) throws IOException
    {
        /*creacion del objeto para leer por teclado*/
        BufferedReader in = new BufferedReader(new InputStreamReader(System.in));
        /*ingreso del tamaño de arreglos*/
        System.out.print("\n Ingrese Numero de Datos a Ingresar : ");
        int tam = Integer.parseInt(in.readLine());
        /*creacion del arreglo*/
        int arr[] = new int[tam];
        System.out.println();
        /*lectura del arreglo*/
        int j = 0;
        for (int i = 0 ; i < arr.length;i++)
```



**PROGRAMA INGENIERIA DE SOFTWARE**  
**Modalidad de Educación a Distancia**  
**MODULO - ESTRUCTURA DE DATOS**



```
{
    j+=1;
    System.out.print("Elemento " + j + " : ");
    arr[i] = Integer.parseInt(in.readLine());
}
burbuja(arr);
}

static void burbuja(int arreglo[])
{
    for(int i = 0; i < arreglo.length - 1; i++)
    {
        for(int j = 0; j < arreglo.length - 1; j++)
        {
            if (arreglo[j] < arreglo[j + 1])
            {
                int tmp = arreglo[j+1];
                arreglo[j+1] = arreglo[j];
                arreglo[j] = tmp;
            }
        }
    }
    for(int i = 0; i < arreglo.length; i++)
    {
        System.out.print(arreglo[i]+"\\n");
    }
}
}
```

Run:

Ingrese Numero de Datos a Ingresar : 6

Elemento 1 : 9  
Elemento 2 : 8  
Elemento 3 : 6  
Elemento 4 : 5  
Elemento 5 : 3  
Elemento 6 : 1  
9



**PROGRAMA INGENIERIA DE SOFTWARE**  
**Modalidad de Educación a Distancia**  
**MODULO - ESTRUCTURA DE DATOS**



8  
6  
5  
3  
1

Ejemplo 2:

```
/*
 * Realizar un programa que permita ordenar una lista de números y de cadenas de
 * caracteres
 */
package ejemplo5;

/**
 *
 * @author Astrid
 */
public class Ejemplo5 {

    /**
     * @param args
     */
    public static void main(String[] args) {

        final int TAMANIO=10;
        int lista[]=new int [TAMANIO];
        rellenarArray(lista);

        String lista_String[]={"Vilma", "Astrid", "Henry", "Marlin", "Angello"};

        System.out.println("Array de números sin ordenar:");
        imprimirArray(lista);

        //ordenamos el array
        intercambio(lista);

        System.out.println("Array de números ordenado:");
        imprimirArray(lista);
    }
}
```





**PROGRAMA INGENIERIA DE SOFTWARE**  
**Modalidad de Educación a Distancia**  
**MODULO - ESTRUCTURA DE DATOS**



```
System.out.println("Array de String sin ordenar:");
imprimirArray(lista_String);

//ordenamos el array
intercambioPalabras(lista_String);

System.out.println("Array de String ordenado:");
imprimirArray(lista_String);

}

public static void imprimirArray (int lista[]){
    for(int i=0;i<lista.length;i++){
        System.out.println(lista[i]);
    }
}

public static void imprimirArray (String lista[]){
    for(int i=0;i<lista.length;i++){
        System.out.println(lista[i]);
    }
}

public static void rellenarArray (int lista[]){
    for(int i=0;i<lista.length;i++){
        lista[i]=numeroAleatorio();
    }
}

private static int numeroAleatorio (){
    return ((int)Math.floor(Math.random()*200));
}

public static void intercambio(int lista[]){

    //Usamos un bucle anidado
    for(int i=0;i<(lista.length-1);i++){
        for(int j=i+1;j<lista.length;j++){
            if(lista[i]>lista[j]){
```



**PROGRAMA INGENIERIA DE SOFTWARE**  
**Modalidad de Educación a Distancia**  
**MODULO - ESTRUCTURA DE DATOS**



```
//Intercambiamos valores
int variableauxiliar=lista[i];
lista[i]=lista[j];
lista[j]=variableauxiliar;

    }
}

}

public static void intercambioPalabras(String lista[]){

    //Usamos un bucle anidado
    for(int i=0;i<(lista.length-1);i++){
        for(int j=i+1;j<lista.length;j++){
            if(lista[i].compareToIgnoreCase(lista[j])>0){
                //Intercambiamos valores
                String variableauxiliar=lista[i];
                lista[i]=lista[j];
                lista[j]=variableauxiliar;

            }
        }
    }
}
```

Run:

Array de números sin ordenar:

126  
185  
51  
187  
146  
0  
125  
139  
9  
11



**PROGRAMA INGENIERIA DE SOFTWARE**  
**Modalidad de Educación a Distancia**  
**MODULO - ESTRUCTURA DE DATOS**



Array de números ordenado:

0  
9  
11  
51  
125  
126  
139  
146  
185  
187

Array de String sin ordenar:

Vilma  
Astrid  
Henry  
Marlin  
Angello

Array de String ordenado:

Angello  
Astrid  
Henry  
Marlin  
Vilma

En entradas anteriores ya aprendimos a ordenar un array numérico con la ordenación de burbuja. En este caso facilitaremos mucho las cosas ya que en la API estándar de JAVA podemos encontrar el método *sort* (Bubble sort) de la clase *Arrays*, que está en el paquete *java.util* que se encargará de ordenar cualquier tipo de array que le pasemos como argumento.

Su uso se verá mejor con el siguiente ejemplo:

/\*

\* Programa que permite ordenar nombres por la ordenación de Burbuja

\* utilizando la API estándar de JAVA llamada: método sort de la clase Arrays

\*/



**PROGRAMA INGENIERIA DE SOFTWARE**  
**Modalidad de Educación a Distancia**  
**MODULO - ESTRUCTURA DE DATOS**



```
package ejemplo7;
import java.util.Arrays;

/**
 *
 * @author Astrid
 */
public class Ejemplo7 {
    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        //Array de String
        //String[] nombres = {"Pepe", "Juan", "Alex","Julian", "Francisco", "Luis"};
        //A este método se le puede pasar cualquier array de cualquier tipo, ya
        //probamos un tipo referencia con el tipo String, así que a continuación
        //podras ver el método sort en acción con un array de tipo primitivo int.
        int[] nombres = {4, 2, 6, -3, 10, 11, 166, 1};
        //Ordena el array
        Arrays.sort(nombres);
        //Mostramos el array ya ordenado
        //for (String i : nombres) {
        for (int i : nombres) {
            System.out.print(i + ", ");
        }
    }
}
```

Run:

-3, 1, 2, 4, 6, 10, 11, 166



PROGRAMA INGENIERIA DE SOFTWARE  
Modalidad de Educación a Distancia  
MODULO - ESTRUCTURA DE DATOS



1.7. Lección 4. ¿Que son los Arreglos Bidimensionales (Matrices)?

Se puede considerar como un vector de vectores. Por tanto es un conjunto de elementos todos del mismo tipo en el que se utilizan dos subíndices para especificar un elemento.

Ej. Una cadena de tiendas está formada por 10 sucursales y cada uno consta de 5 secciones (Lácteos/Bebidas/... /carnes). En la siguiente tabla o matriz (matemático) se representan las ventas mensuales en soles.

Una matriz es una estructura de dos dimensiones: horizontal (**filas**) y vertical (**columnas**), que contiene elementos del mismo tipo. Se puede hacer referencia a cada uno de los elementos (**celdas**) mediante un **índice** de fila y otro de columna.

Por ejemplo, la tabla siguiente (*temperaturas*) muestra las temperaturas máxima y mínima a lo largo de una semana (las filas indican el día de la semana y las columnas las temperaturas mínima y máxima, respectivamente. Por ejemplo, la temperatura mínima del 4º día es: temperaturas (3, 0) = 5 grados).

temperaturas

	0	1
0	7	15
1	8	17
2	6	13
3	5	14
4	7	14
5	6	16
6	5	13

Tabla 1.7. Temperaturas a lo largo de una semana

... Piense en algunos ejemplos de arreglos bidimensionales:

- Las notas de 3 periodos académicos de cada uno de los 25 estudiantes de Estructura de datos.
- El inventario de los 400 medicamentos diferentes en sus tres tipos de presentaciones diferentes presentaciones.
- Los puntos obtenidos por 5 candidatas a un concurso de belleza en dos presentaciones (Ropa de noche y Preguntas de cultura general)



**PROGRAMA INGENIERIA DE SOFTWARE**  
**Modalidad de Educación a Distancia**  
**MODULO - ESTRUCTURA DE DATOS**



Los lenguajes de programación permiten utilizar este tipo de estructuras. Para esto en Java se emplea la siguiente sintaxis:

- Declaración de variables de tipo matriz:

< tipo de datos de los elementos > [ ] [ ] <nombre de la variable>;

Ejemplo: *int [ ] [ ] temperaturas;*

- Acceso:

o Al conjunto. Por ejemplo asignar una matriz a otra del mismo tipo:  
matriz2 = matriz1.

Como en el caso de los vectores, esta instrucción no copia el contenido de matriz1 en matriz2, sino que hace que matriz2 apunte a la misma posición de memoria que matriz1.

o A un elemento de la matriz:  
<variable\_tipo\_matriz>[<índice1>] [<índice2>];

(Con lo que se podrá realizar cualquier operación acorde con el tipo del elemento correspondiente). Por ejemplo: *temperaturas [3] [1] = temperaturas [3] [1] + 2;*

**Ejemplo.**

El siguiente código permite encontrar la temperatura mínima de la semana y el día en que se produjo la máxima diferencia (en caso de coincidencia de varias se muestra el primero de ellos). En el ejemplo anterior los resultados serían:

- Temperatura mínima: 5 grados.
- Día de máxima diferencia de temperaturas: 5 (10 grados).

```
import java.io.*;
public class PruebaMatrices {
    static BufferedReader linea=new BufferedReader(new
    InputStreamReader(System.in));
    public static void leerMatriz (int [][] temperaturas) throws
    NumberFormatException, IOException{
        int i,j;
        for (i = 0;i < 7; i ++){
            for (j = 0; j < 2; j ++){
```



**PROGRAMA INGENIERIA DE SOFTWARE**  
**Modalidad de Educación a Distancia**  
**MODULO - ESTRUCTURA DE DATOS**



Universidad  
de Cartagena  
Fundada en 1827



Accreditación Institucional de Alta Calidad  
Resolución 2583 del 26 de febrero de 2014. Ministerio de Educación Nacional

```
        System.out.println ("Valor dia: " + i + " extremo: " + j + ": ");
        temperaturas [i] [j] = Integer.parseInt (linea.readLine ());
    }
}

public static int min (int [][] temperaturas) {
    int resul, i;
    resul = temperaturas [0] [0];
    for (i = 1; i < 7; i++)
        if (temperaturas [i] [0] < resul)
            resul = temperaturas [i] [0];
    return resul;
}

public static int maxDif (int [] [] temperaturas) {
    int resul, i, dif;
    resul = 0;
    dif = temperaturas [0][1]- temperaturas [0][0];
    for (i = 1; i < 7; i++) {
        if (temperaturas [i][1] - temperaturas [i][0] > dif) { dif =
            temperaturas [i][1] - temperaturas [i][0]; resul = i;
        }
    }
    return resul;
}

public static void main(String[] args) throws NumberFormatException,
IOException{
    int [][] temperaturas = new int [7][2];
    int minimaTemperatura, diferenciaTemperaturas;
    leerMatriz (temperaturas); minimaTemperatura =
    min (temperaturas); diferenciaTemperaturas =
    maxDif (temperaturas);    System.out.println
    ("Resultados:");
    System.out.println ("Temperatura minima: " + minimaTemperatura);
    System.out.println ("Dia extremo: " + diferenciaTemperaturas);
}
}
```





**PROGRAMA INGENIERIA DE SOFTWARE**  
**Modalidad de Educación a Distancia**  
**MODULO - ESTRUCTURA DE DATOS**



#### 1.4.2.1.3. N-dimensionales.

Por extensión, lo explicado para una y dos dimensiones se puede aplicar al caso de matrices *N*-dimensionales.

#### 1.4.2.2. Estructuras de datos heterogéneas.

Como ya se ha indicado, están constituidas por un conjunto de tipos de datos (ya sean datos simples u otras estructuras de datos) de diferente naturaleza. La forma básica de estructura de datos heterogénea es el **registro** cuyos elementos se llaman **campos** (o **atributos**)<sup>14</sup>.

Por ejemplo, la figura siguiente muestra un registro (*alumno*) cuyos campos son: el número de matrícula (*numeroMatricula*), apellidos (*apellidos*), nombre (*nombre*), dirección de correo electrónico (*eMail*), año de nacimiento (*año*) y calificación (*calificacion*).

alumno

bc2658	Sánchez Arellano	Estela	esanchez@servidor.es	1987	6.75
numeroMatricula	apellidos	nombre	eMail	año	calificacion

Tabla 1.8. Registro alumno

Para manejar este tipo de estructuras en Java<sup>15</sup>, se construye una clase dentro de la cual se definen los datos que van a formar parte de la estructura, así como uno o varios constructores, como se puede ver en el siguiente ejemplo (correspondiente a la figura):

```
class RegistroAlumno {  
  
    public String numeroMatricula;  
    public String apellidos;  
    public String nombre;  
    public String eMail; public  
    int año;  
}
```



<sup>14</sup> Con frecuencia uno de los campos (o combinación de ellos) se utiliza como identificativo del registro. A dicho campo (o conjunto) se le denomina **clave** (key).

<sup>15</sup> Existen otras operaciones. Consultar el manual del lenguaje.



**PROGRAMA INGENIERIA DE SOFTWARE**  
**Modalidad de Educación a Distancia**  
**MODULO - ESTRUCTURA DE DATOS**



```
public float calificacion;
```

```
public RegistroAlumno (){\n    numeroMatricula= null;\n    apellidos = null; nombre\n    = null;\n    eMail= null;\n    año = 1980;\n    calificacion = 0;\n}
```



**PROGRAMA INGENIERIA DE SOFTWARE**  
**Modalidad de Educación a Distancia**  
**MODULO - ESTRUCTURA DE DATOS**



- Acceso:

o Al conjunto. Por ejemplo apuntar con una variable de tipo registro mismo sitio que apunta otra del mismo tipo:

variable2\_RegistroAlumno = variable1\_RegistroAlumno;

o A un campo del registro:

<variable>.<campo>;

Con lo que se podrá realizar cualquier operación acorde con el tipo del elemento correspondiente. Por ejemplo: *alumno.eMail* = *"esanchez@servidor.es"*;

Normalmente, el trabajo con un registro (o con un conjunto pequeño de variables independientes de tipo registro) ofrece "poco juego". Lo habitual es utilizar una "colección" de registros estructurados en un vector, o mucho mejor, almacenarlos, como un "**Archivo**" en un dispositivo de almacenamiento externo (típicamente disco).

La utilización de Archivos en dispositivos de almacenamiento externos se explica en el apartado 1.5. A continuación se muestra un ejemplo que utiliza un vector de registros del tipo anterior cuyo modelo se ilustra gráficamente en la figura siguiente:

	numeroMatricula	apellidos	nombre	eMail	año	calificacion
0	aa1253	Arias González	Felipe	farias@servidor.es	1988	3.50
1	ax0074	García Sacedón	Manuel	mgarcia@servidor.es	1985	8.35
2	mj7726	López Medina	Margarita	mlopez@servidor.es	1990	7,70
3	lp1523	Ramírez Heredia	Jorge	jramirez@servidor.es	1998	4,50
4	bc2658	Sánchez Arellano	Estela	esanchez@servidor.es	1989	6.75
5	gb1305	Yuste Peláez	Juan	jyuste@servidor.es	1990	5,50

*Tabla 1.9. Vector de registros.*

El programa que se muestra a continuación se ha incluido en dos Archivos (clases) que forman parte del mismo paquete (*package*):

- *RegistroAlumno*, que contiene el constructor del registro (construye un registro vacío), así como los métodos *aCadena*, que devuelve el contenido del



**PROGRAMA INGENIERIA DE SOFTWARE**  
**Modalidad de Educación a Distancia**  
**MODULO - ESTRUCTURA DE DATOS**



registro en un *String*, y *cargarRegistro*, que introduce los datos recogidos por teclado en un registro.

- *PruebaRegistro*, en el que aparece el programa principal e incluye dos métodos, *cargarTabla*, que permite cargar la estructura en la memoria central y *mediaCalif*, que utiliza la estructura anterior para calcular la calificación media.

Primero se muestra la clase *RegistroAlumno*:

```
Import java.io.*;
class RegistroAlumno {
    public RegistroAlumno () {
        numeroMatricula=    null;
        apellidos = null;
        nombre = null;
        eMail= null; año
        = 1980;
        calificacion = 0;
    }
    public String aCadena () {
        return numeroMatricula + " " + apellidos + " " + nombre + " " + eMail + " " +
        año + " " + calificacion;
    }
    public String numeroMatricula;
    public String apellidos;
    public String nombre;
    public String eMail; public
    int año;
    public float calificacion;
    public void cargarRegistro () throws IOException {
        BufferedReader linea = new BufferedReader (new InputStreamReader (System.in));
        System.out.println ("Numero de matricula: ");
        numeroMatricula = new String (linea.readLine ());
        System.out.println ("Apellidos: ");
        apellidos = new String (linea.readLine ());
        System.out.println ("Nombre: ");
        nombre = new String (linea.readLine ());
        System.out.println ("Correo electronico: ");
        eMail = new String (linea.readLine ());
        System.out.println ("Año de nacimiento: "); año
        = Integer.parseInt (linea.readLine());
        System.out.println ("Calificación: ");
```



**PROGRAMA INGENIERIA DE SOFTWARE**  
**Modalidad de Educación a Distancia**  
**MODULO - ESTRUCTURA DE DATOS**



```
calificacion = Float.parseFloat (linea.readLine());
System.out.println (this.aCadena ());
}
}
```

Como se ha visto en el apartado 1.2.4., los métodos *aCadena* y *cargarRegistro* son métodos de objeto (sin modificador *static*), y para utilizarlos desde fuera de la clase tendrá que hacerse de la forma *<nombreVariable>.<nombreMétodo>*. Para poder utilizar los métodos de objeto, es necesario que previamente hayamos utilizado el constructor del objeto sobre la variable correspondiente, como hacemos con las instrucciones:

```
for (i = 0; i < 6;i++)
    alumnos [i]= new RegistroAlumno ();
```

A continuación aparece la clase *PruebaRegistro*, donde se incluye el programa principal:

```
import java.io.*;
public class PruebaRegistro {
    static void cargarTabla (RegistroAlumno [ ] alumnos) throws IOException {
        int i;
        for (i = 0; i < 6; i++) {
            System.out.println ("Datos del alumno N: "+ i);
            alumnos [i].cargarRegistro ();
        }
    }
    static float mediaCalif (RegistroAlumno [ ] alumnos) {
        float resul = 0;
        int i;
        for (i = 0; i < 6; i++) {
            System.out.println(alumnos [i].aCadena ());
            resul = resul + alumnos [i].calificacion;
        }
        return resul/6;
    }
    public static void main (String [ ] args) throws IOException {
        RegistroAlumno [ ] alumnos = new RegistroAlumno [6];
        float media;
        int i;
        for (i = 0; i < 6; i++)
```



**PROGRAMA INGENIERIA DE SOFTWARE**  
**Modalidad de Educación a Distancia**  
**MODULO - ESTRUCTURA DE DATOS**



```
    alumnos [i]= new RegistroAlumno ();  
    cargarTabla (alumnos);  
    media = mediaCalif (alumnos);  
    System.out.println ("La media de las calificaciones es: " + media);  
}  
}
```



## **REFERENCIAS BIBLIOGRÁFICAS**



A continuación algunas *referencias bibliográficas* sobre este tema.

### **BIBLIOGRAFIA**

#### **FUENTES DOCUMENTALES**

- ✿ INSUASTY R, Luis Delfín, Guía "A","B","C","D" de aprendizaje autónomo. Bogotá Colombia, Unad- Cafan
- ✿ MAURREN, Priestley . Técnicas y estrategias del pensamiento crítico. México D.F. 1996 (reimp .2000). Trillas.
- ✿ ARCEO B, Frida y Otro. Estrategias Decentes Para un Aprendizaje Significativo. Mexico D,F 1999. McGraw-HILL
- ✿ KENNETH C, louden . Lenguajes de programación (segunda edición). México D.F 2004. Thompson
- ✿ AGUILAR, Luis. Fundamentos de programación, algoritmos, estructura de datos y Objetos (tercera edición). España. 2003. McGRAW-HILL.
- ✿ AGUILAR, Luis. Programación en C++, Algoritmos, estructura de datos y Objetos España. 2000. McGRAW-HILL.
- ✿ DEYTEL Y DEYTEL. Como programar C++ (segunda Edición). México D.F. 1999. Prentice Hall. McGRAW-HILL
- ✿ FARREL, Joyce, introducción a la programación lógica y diseño. México D.F 2000. Thompson

#### **Sitios Web**

- ✿ [http://www.geocities.com/david\\_ees/Algoritmia/curso.htm](http://www.geocities.com/david_ees/Algoritmia/curso.htm) (Curso de algoritmia)
- ✿ <http://www.ilustrados.com/publicaciones/EpZVVEZpyEdFpAKxjH.php> (Lenguajes de Programación)
- ✿ <http://www.ilustrados.com/buscar.php> (Algoritmos)
- ✿ <http://www.inf.utfsm.cl/~mcloud/iwi-131/diapositivas.html> (Algoritmos)
- ✿ <http://www.ucsm.edu.pe/rabarcaf/vonuep00.htm> (Diccionario académico)
- ✿ <http://www.funlam.edu.co/bired/index.asp-offset=0.htm> (Aprendizaje Autónomo)



